



UNIVERSITAT POLITÈCNICA
DE CATALUNYA
BARCELONATECH

FACULTAT D'INFORMÀTICA DE BARCELONA (FIB)

MASTER IN INNOVATION AND RESEARCH IN INFORMATICS

Development of an oceanographic application in HPC

Advisor

Eng. Filippo Mantovani

Ponent

Prof. Jordi Domingo Pascual

Student

Davide Basciano
Matr. M63000429

Abstract

High Performance Computing (HPC) is used for running advanced application programs efficiently, reliably, and quickly.

In earlier decades, performance analysis of HPC applications was evaluated based on speed, scalability of threads, memory hierarchy. Now, it is essential to consider the energy or the power consumed by the system while executing an application.

In fact, the High Power Consumption (HPC) is one of biggest problems for the High Performance Computing (HPC) community and one of the major obstacles for exascale systems design.

The new generations of HPC systems intend to achieve exaflop performances and will demand even more energy to processing and cooling. Nowadays, the growth of HPC systems is limited by energy issues

Recently, many research centers have focused the attention on doing an automatic tuning of HPC applications which require a wide study of HPC applications in terms of power efficiency.

In this context, this paper aims to propose the study of an oceanographic application, named OceanVar, that implements Domain Decomposition based 4D Variational model (DD-4DVar), one of the most commonly used HPC applications, going to evaluate not only the classic aspects of performance but also aspects related to power efficiency in different case of studies.

These work were realized at Bsc (Barcelona Supercomputing Center), Spain within the Mont-Blanc project, performing the test first on HCA server with Intel technology and

then on a mini-cluster Thunder with ARM technology.

In this work of thesis it was initially explained the concept of assimilation date, the context in which it is developed, and a brief description of the mathematical model 4DVAR.

After this problem's close examination, it was performed a porting from Matlab description of the problem of data-assimilation to its sequential version in C language. Secondly, after identifying the most onerous computational kernels in order of time, it has been developed a parallel version of the application with a parallel multiprocessor programming style, using the MPI (Message Passing Interface) protocol.

The experiments results, in terms of performance, have shown that, in the case of running on HCA server, an Intel architecture, values of efficiency of the two most onerous functions obtained, growing the number of process, are approximately equal to 80%.

In the case of running on ARM architecture, specifically on Thunder mini-cluster, instead, the trend obtained is labeled as "SuperLinear Speedup" and, in our case, it can be explained by a more efficient use of resources (cache memory access) compared with the sequential case.

In the second part of this paper was presented an analysis of the some issues of this application that has impact in the energy efficiency.

After a brief discussion about the energy consumption characteristics of the Thunder chip in technological landscape, through the use of a power consumption detector, the Yokogawa Power Meter, values of energy consumption of mini-cluster Thunder were evaluated in order to determine an overview on the power-to-solution of this application to use as the basic standard for successive analysis with other parallel styles.

Finally, a comprehensive performance evaluation, targeted to estimate the goodness of MPI parallelization, is conducted using a suitable performance tool named Paraver, developed by BSC.

Paraver is such a performance analysis and visualisation tool which can be used to analyse MPI, threaded or mixed mode programmes and represents the key to perform a

parallel profiling and to optimise the code for High Performance Computing.

A set of graphical representation of these statistics make it easy for a developer to identify performance problems. Some of the problems that can be easily identified are load imbalanced decompositions, excessive communication overheads and poor average floating operations per second achieved.

Paraver can also report statistics based on hardware counters, which are provided by the underlying hardware.

This project aimed to use Paraver configuration files to allow certain metrics to be analysed for this application.

To explain in some way the performance trend obtained in the case of analysis on the mini-cluster Thunder, the tracks were extracted from various case of studies and the results achieved is what expected, that is a drastic drop of cache misses by the case ppn (process per node) = 1 to case ppn = 16.

This in some way explains a more efficient use of cluster resources with an increase of the number of processes.

Indice

Abstract	2
Indice.....	5
Introduction.....	6
State of the art	9
Chapter 1: Model OceanVar	13
Paragraph 1.1: Linearization and Preconditioning	14
Chapter 2: Software DD-4DVAR	15
Paragraph 2.1: Description DD-4DVAR sequential application	15
Paragraph 2.2: Profiling DD-4DVAR sequential	20
Chapter 3: Parallel version DD4DVAR.....	25
Paragraph 3.1: Mont-Blanc Project.....	25
Paragraph 3.1.1: Environment of application's development.....	27
Paragraph 3.2: MPI Implementation.....	27
Paragraph 3.3: Results.....	30
Paragraph 3.3.1: Analysis in HCA.....	31
Paragraph 3.3.2: Analysis in Thunderx cluster	35
Chapter 4: Power Efficiency	40
Paragraph 4.1: Supercomputer performances, Power Consumption and rank lists	40
Paragraph 4.2: Research in Exascale Systems	42
Paragraph 4.4: Power Trace of the application	43
Chapter 5: Inside Paraver Tool	47
Paragraph 5.1: Extrae	48
Paragraph 5.1.1: Extraction process of a trace.....	49
Paragraph 5.2: Analysis with Paraver	51
Paragraph 5.2.1: Parallel version with ppn=4	51
Paragraph 5.2.2: Parallel version with ppn=16	58
Paragraph 5.2.3: Vertical deepening of the cache management	60
Conclusioni e sviluppi futuri.....	65
Bibliografia	67

Introduction

The High Performance Computing (HPC) plays an important role in scientific research to solve complex computational problems in meteorology, astrophysics and geophysics. Today, computational sciences are able to address the most complex scientific problems and investigate phenomena unimaginable even ten years ago.

This is possible thanks to the increase in performance that has characterized the computer systems in recent years.

However, this rising demand for high-performance technology is bringing to light a significant factor that is the need for a higher and higher energy consumption.

The trend of recent years shows that many of the HPC sector companies dedicate more attention to the energy efficiency issue in its own research centers, in order to reduce the energy consumption and the consequent environmental impact. For this reason, they introduce as new metric of evaluation the **power efficiency** estimated as ***Flops / W***, that is, the number of floating point operations per second per watt.

In this optic, an important role is playing the *European Project Mont Blanc*, at the headquarters of the Barcelona Supercomputing Center (BSC). It aims to the design and construction of a new high-end HPC platform, capable of providing a new value of the performance / energy ratio in the execution of scientific applications.

The following thesis work, realized thanks to the support and cooperation with the above-mentioned project, focuses attention on the concept of data assimilation.

Historically the assimilation data has been developed for the analysis of the two main terrestrial ecosystems, that is the one generated by the oceans and that generated from the atmosphere.

Regardless of the specific application, the entire process of analysis and prediction consists essentially of three phases:

- **collection of observed data**

Observed data can be of various types: measurements from buoys and ships, measurements from satellites, etc.

- **determination of the initial condition (initial condition)**

Is necessary to find the most accurate initial state, considering all the observed data collected and available in a fixed time window.

- **determination of the instant current solution (status "current") through the Forecasting Model**

by solving the equations of the physical-mathematical model, known initial state and any boundary conditions.

The observed data may be of various types: ground stations (SYNOP), radiosonde (TEMP), boe (BUOYS), ships, measurements from commercial aircraft and various types of satellite measurements.

In the case of oceanographic forecasts, the observations may include:

- data of the sea level anomaly
- vertical temperature profiles obtained from probes called XBT (Expendable BathyTermograph)
- acquisition and development of temperature and salinity profiles provided by the system of boe

The goal shared by the scientific community that deals with data assimilation is, in short, the opportunity to integrate the experimentally acquired data with those supplied by mathematical models in order to make them more similar as possible, in order to improve the knowledge of the surrounding ecosystem.

It is obvious that the benefit that derives from this synergy stimulate the development of

highly efficient and reliable algorithms and software for the data assimilation , able to cope both with a growing amount of data both with sensibility to errors that inevitably disturb data and models.

The software tool used in the analysis of the problem is the *OceanVar* software (Ocean Variational Data Assimilation) developed by CMCC (Euro-Mediterranean Centre for Climate Change).

The study of this application, in this work of thesis, was articulated in the following phases:

- Software analysis, initially released in Matlab version, and development in C language of a first purely sequential version.
- Study of the software in order to identify the most onerous computational kernels.
- Identified the interested code portions, realization of a parallel version by use of the MPI programming style
- Instrumentation of the code and the study of the parallel trace with the use of the tool *Paraver*.
- Considerations on the energy consumption of the application in the various test cases.

State of the art

A first approach to solving the problem Assimilation data was suggested by George P. Cressman (1959) said Cressman analysis.

A first approach to solving the problem Assimilation data was suggested by George P. Cressman (1959) said Cressman analysis. The method provides as an approximation of the initial condition the point obtained by performing linear interpolation points $\{(a_i, b_i)\}_{i=1,2,\dots,n}$ defined in such a way that the abscissas a_i represent the coordinates of the location of the discrete domain in which is defined the forecasting model solution and the ordinates b_i are calculated as follows.

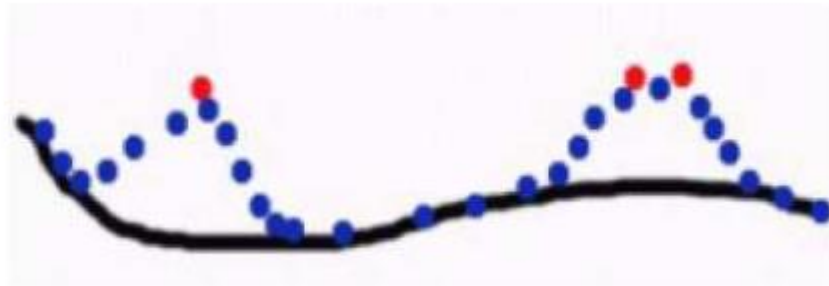
It defines a sphere of center equal to the observed data and the radius R said "sphere of influence" of the data observed on the forecast model solution (if the distance between the points in which there are observations and the points in which there is defined the solution of the model is less than R , it is assumed that there is an influence of the data observed on the model) and define the values such that:

- b_i coincides with the data value observed if in a_i this is present
- b_i coincides with the value of the forecast model solution if a_i is far from points which are present data observed of a greater segment of R
- b_i coincides with the difference between the forecast model solution and observed data if a_i is far from points which are present in an observed data of a minor segment of R

In the following figure, the black line represents the short-term prediction of the prediction model and the red dots the observations

The Cressman Analysis method produces the line in blue dots, which coincides with the prediction of the model at points distant from the observations, and relaxes to the observed data where they are present.

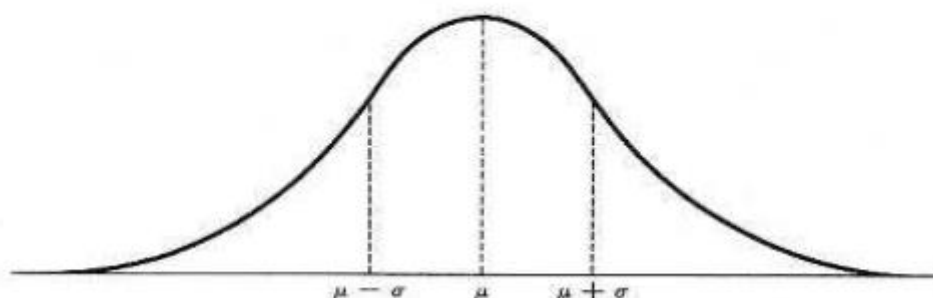
In this method lacks the quality control of data and especially are not taken into consideration the weights of the errors of the observed data and the model.



Among the methods used there is the Optimal Interpolation.

It is a method of approximation in the least squares sense. This method allows to make the first considerations on the weight that error estimate on the data must have in the calculation of the solution provided by data assimilation.

The error on the data is estimated by calculating the variance of the set values that describe the data. In fact, assumed that the data are affected by errors distributed according to a Gaussian, the variance σ , for definition is the distance between the mean value of the set of data μ and the inflection point of the curve.



If we assume as "more reliable" data the average value, we have that the variance, which represents the "deviation" of the remaining data from the medium value, it gives us an estimate of the "deviation" of the data from the "more trusted " value, which is by definition the error.

From such method were conducted studies on how to select the observed data trying to consider only those with high reliability beginning to look for ways to involve the forecasting model to produce a more accurate estimate of the error on the data produced by the short-prediction term.

The main methods of data assimilation problem resolution including the issues on the error weights in the data are essentially attributable to four models (variational model 3D **3Dvar** and 4D **4DVar**, Kalman filter **KF** and Ensemble Kalman Filter **EnKF**) which differ in the presence or absence of the variable time and for the type of mathematical problem solving.

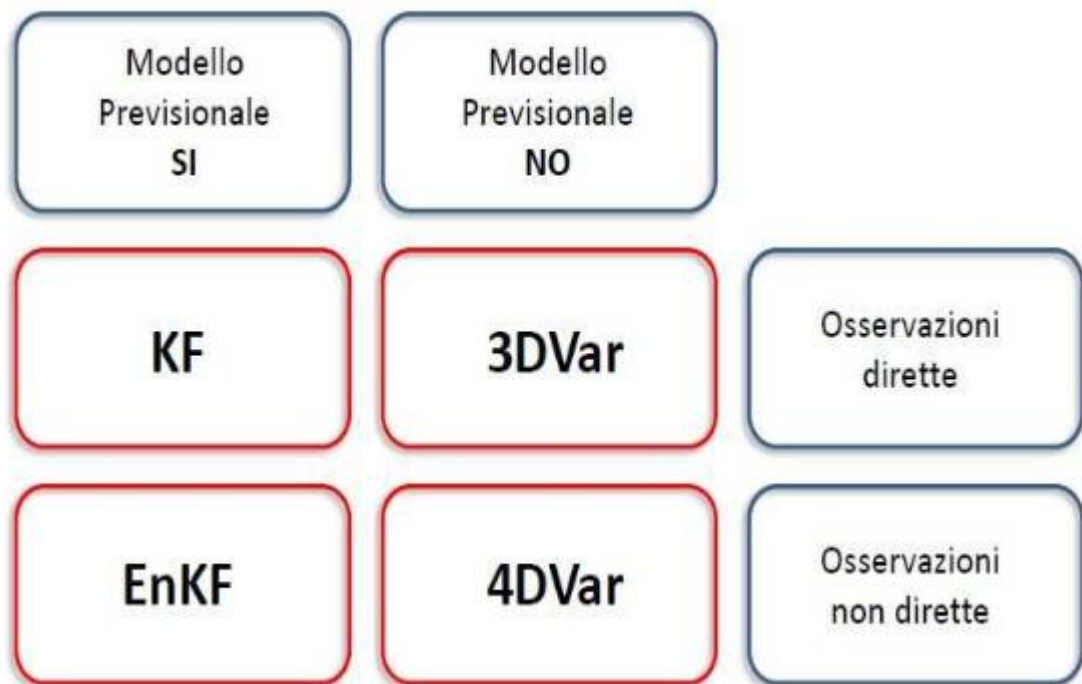
Is possible to observe, for example, that the presence of the time variable in the models Data Assimilation and then the tendency to use the 4D type models appears only in recent years dictated by the increasing availability of data from satellites and then by the introduction of a functional "more complex" than the interpolant functional used for direct observations.

Today, there isn't a tendency that pushes the use of a method rather than another, it is still in phase of comparison between any advantages or disadvantages that the choice of a method can bring.

There are no criteria in literature that dictate the choice of the resolution method.

We have observed, in that regard, that the choice of the resolution method essentially depends on two factors summarized in the following figure:

1. Type of observed data (available and / or chosen) that determines the functional;
2. Availability of a software that implements the forecasting model. This possibility allows a more accurate estimate of the error on the data produced by the short-term prediction of the model.



Chapter 1: Model OceanVar

This chapter describes the mathematical model implemented by OceanVar software developed and used by CMCC for the assimilation of data in the study of evolution of the Mediterranean Sea.

The scheme upon which the Software OceanVar is based on is of type *Domain Decomposition based 4D Variational model (DD-4DVar)*.

4DVAR, according to the name, is a four dimensional variational method and is actually a direct generalization of 3D-VAR to handle observations that are distributed in time.

The cost function is the same, provided that the observation operators are generalized to include a forecast model that will allow a comparison between the model state and the observations at the appropriate time.

4DVAR seeks the *initial condition* such that the forecast best fits the observations within the assimilation interval

It is described in this way, let:

- N = number of time steps
- $x_0 = x(t_0)$ vector from forecasting model
- $y_0 = y(t_0)$, $y_1 = y(t_1)$, $y_N = y(t_N)$ observation vectors

the characteristic equation of the model that calculates the minimum of the function is:

$$J(x^{DA}) = \frac{1}{2}(x - x^{DA})^T B^{-1} (x - x^{DA}) + \frac{1}{2} \sum_{k=0}^N (y_k - G_k x^{DA})^T R^{-1} (y_k - G_k x^{DA}) + \frac{1}{2} \sum_{k=0}^N (M^k x^{DA+} - M^k x^{DA-})^T (M^k x^{DA+} - M^k x^{DA-})$$

Where: $G_k = H_k M^k$

State variable: $x_k = [T, S, n, u, v]^T$

- T : three-dimensional temperature field
- S : three-dimensional salinity field
- n : two-dimensional free surface elevation
- u,v : horizontal velocity components

Observation: Fixed a time step Δt for temporal discretization and assumed

$t_{i+1} = t_i + \Delta t$ for all $t_i = t_0, \dots, t_n$ we have that:

$$x_{t1} = M x_{t1+\Delta t}$$

Paragraph 1.1: Linearization and Preconditioning

Let:

- $x = x_0 + (x - x_0) = x_0 + \delta x$ linearization
- $B = VV^T$ preconditioning
- $v = V^T \delta x$ change of variable
- $d_k = y_k - G_k x_0$ misfit
- $G_k = H_k M_k$ forecasting model

we have the preconditioned DD-4DVAR function:

$$J(v) = \frac{1}{2} v^T v + \frac{1}{2} \sum_{k=0}^N (H_k M^k V_v - d_k)^T R^{-1} (H_k M^k V_v - d_k) + \frac{1}{2} \sum_{k=0}^N (M^k V_v^+ - M^k V_v^-)^T (M^k V_v^+ - M^k V_v^-)$$

The gradient of $J(v)$ is given in:

$$\nabla J(v) = v + \frac{1}{2} \sum_{k=0}^N V^T (M^T)^k H_k^T R^{-1} (H_k M^k V_v - d_k)$$

The $J(v)$ function is minimized using the L-BFGSB method (a quasi-Newton method).

Chapter 2: Software DD-4DVAR

In the following work of thesis has been analyzed and studied in detail the DD4DVAR software, developed by INGV (Bologna Institute of Geophysics and Volcanology) and used by CMCC (Euro-Mediterranean Centre for Climate Change), in order to contribute, on first analysis, to its writing in a sequential version of C language, and in second analysis, to its optimization in a parallel context.

Paragraph 2.1: Description DD-4DVAR sequential application

The under consideration software consists of a main library named *Subdomain.c* which in turn is structured in five main modules.

In *Subdomain.c* library were, in the first instance, loaded the input data related to the measures made,

These data, present initially in .mat format, denominated "ShallowWaterStateN64", were converted to .csv format (comma separated value) using a greater precision of significant digits in order to be able to perform with the accuracy required subsequent matrix calculations to stored, ultimately, in suitable matrix data structures.

Subsequently are built matrices of observations containing all zero values except some values of the initial dataset placed in specific locations calculated with random choice.

At this point, the library performs the *BuildModel.c* module for the construction of the work grid, in essence, the forward and backward models $Me M^T$ defined previously.

The relative pseudocode is the following:

```

1: function buildmodels( $n_x, n_y, x_b$ )
2:    $n \leftarrow n_x * n_y$ 
3:    $h \leftarrow \text{reshape}(x_b(1 : n), n_x, n_y)$ 
4:    $u \leftarrow \text{reshape}(x_b(n + 1 : 2 * n), n_x, n_y)$ 
5:    $v \leftarrow \text{reshape}(x_b(2 * n + 1 : 3 * n), n_x, n_y)$ 
6:                                     ▷ Define the nine components of  $A$  on all the points of subdomain
7:    $A(:, :, 1) \leftarrow 0$ 
8:    $A(:, :, 2) \leftarrow 1$ 
9:    $A(:, :, 3) \leftarrow 0$ 
10:   $A(:, :, 4) \leftarrow g. * h - u.^2$ 
11:   $A(:, :, 5) \leftarrow 2. * u$ 
12:   $A(:, :, 6) \leftarrow 0$ 
13:   $A(:, :, 7) \leftarrow -1. * u. * v$ 
14:   $A(:, :, 8) \leftarrow v$ 
15:   $A(:, :, 9) \leftarrow u$ 
16:                                     ▷ Define the nine components of  $B$  on all the points of subdomain
17:   $B(:, :, 1) \leftarrow 0$ 
18:   $B(:, :, 2) \leftarrow 0$ 
19:   $B(:, :, 3) \leftarrow 1$ 
20:   $B(:, :, 4) \leftarrow A(:, :, 7)$ 
21:   $B(:, :, 5) \leftarrow A(:, :, 8)$ 
22:   $B(:, :, 6) \leftarrow A(:, :, 9)$ 
23:   $B(:, :, 7) \leftarrow g. * h - v.^2$ 
24:   $B(:, :, 8) \leftarrow 0$ 
25:   $B(:, :, 9) \leftarrow 2. * v$ 
26:  return  $A, B$ 
27: end function

```

The next phase presents a series of matrix calculations to define the preliminary data structures for the following modules, such as the definition of the misfit of the functional $J_{\text{misf}}(v)$ and the definition of the starting point v_0 .

Concerning the *TSVD (Truncated Singular Value Decomposition)* function, included in the Matlab code with the name *svds*, the idea was to use a standard software library for numerical linear algebra named LAPACK (Linear Algebra Package), specifically a two-level C interface named LAPACKE.

Considering that the researchers in this field have been unable to get it working in the right way so, because of the lack of technical support to resolve this issue, it was solved temporarily by loading a file.mat of data structures of interest.

Once calculated the necessary data structures to pass in input, it is executed the module *ComputeJ.c* for the calculation of the functional $J(v)$ and the module *ComputeGradJ.c* for the calculation of gradient $\nabla J(v)$ as described in the previous model.

The various instructions of these two listings were therefore executed:

```

1: function  $J(p_x, p_y, pid_x, pid_y, n_x, n_y, dx, dy, v, N, H, R, V, A, B, V^{west}, V^{right}, V^{south}, V^{north}, \{d_k\}_{k=0, \dots, N})$ 
2:    $J \leftarrow v v^T$ 
3:   for  $i = 0$  to  $N$  do                                ▷ Update the value of  $J$  with the contribution related to my subdomain
4:      $x_k \leftarrow V v$ 
5:      $y_k \leftarrow applytheforwardmodel(A, B, x_k, n_x, n_y, dx, dy, k)$ 
6:      $z_k \leftarrow H y_k$ 
7:      $z_k \leftarrow z_k - d_k$ 
8:      $z_k \leftarrow R^{-1} z_k$ 
9:      $J \leftarrow J + z_k^T z_k$ 
10:  end for
11:  if  $pid_x > 0$  then                                ▷ Update the value of  $J$  with the contribution related to the west processor subdomain
12:     $v^{west} \leftarrow extractboundary(v, 'west', 1, n_y)$ 
13:     $x_k^{west} \leftarrow V^{west} v^{west}$ 
14:     $yr_k^{west} \leftarrow sendreceive(pid_x - 1, pid_y, y_k^{west})$ 
15:     $y_k^{west} \leftarrow y_k^{west} - yr_k^{west}$ 
16:     $J \leftarrow J + y_k^{west T} y_k^{west}$ 
17:  end if
18:  if  $pid_x < p_x$  then                                ▷ Update the value of  $J$  with the contribution related to the east processor subdomain
19:     $v^{east} \leftarrow extractboundary(v, 'east', 1, n_y)$ 
20:     $x_k^{east} \leftarrow V^{east} v^{east}$ 
21:     $yr_k^{east} \leftarrow sendreceive(pid_x + 1, pid_y, y_k^{east})$ 
22:     $y_k^{east} \leftarrow y_k^{east} - yr_k^{east}$ 
23:     $J \leftarrow J + y_k^{east T} y_k^{east}$ 
24:  end if
25:  if  $pid_y < p_y$  then                                ▷ Update the value of  $J$  with the contribution related to the south processor subdomain
26:     $v^{south} \leftarrow extractboundary(v, 'south', n_x, 1)$ 
27:     $x_k^{south} \leftarrow V^{south} v^{south}$ 
28:     $yr_k^{south} \leftarrow sendreceive(pid_x, pid_y + 1, y_k^{south})$ 
29:     $y_k^{south} \leftarrow y_k^{south} - yr_k^{south}$ 
30:     $J \leftarrow J + y_k^{south T} y_k^{south}$ 
31:  end if
32:  if  $pid_y > 0$  then                                ▷ Update the value of  $J$  with the contribution related to the north processor subdomain
33:     $v^{north} \leftarrow extractboundary(v, 'north', n_x, 1)$ 
34:     $x_k^{north} \leftarrow V^{north} v^{north}$ 
35:     $yr_k^{north} \leftarrow sendreceive(pid_x, pid_y - 1, y_k^{north})$ 
36:     $y_k^{north} \leftarrow y_k^{north} - yr_k^{north}$ 
37:     $J \leftarrow J + y_k^{north T} y_k^{north}$ 
38:  end if
39:   $J \leftarrow \frac{1}{2} J$ 
40:  return  $J$ 

```

```

1: function  $J(n_x, n_y, dx, dy, v, N, H, R, V, VT, A, B, \{d_k\}_{k=0, \dots, N})$ 
2:    $gradJ \leftarrow v$ 
3:   for  $i = 0$  to  $N$  do
4:      $x_k \leftarrow V v$ 
5:      $x_k \leftarrow applytheforwardmodel(A, B, x_k, n_x, n_y, dx, dy, k)$ 
6:      $x_k \leftarrow H x_k$ 
7:      $x_k \leftarrow x_k - d_k$ 
8:      $x_k \leftarrow R^{-1} x_k$ 
9:      $x_k \leftarrow H x_k$ 
10:     $x_k \leftarrow applythebackwardmodel(A, B, x_k, n_x, n_y, dx, dy, k)$ 
11:     $gradJ \leftarrow gradJ + VT x_k$ 
12:  end for
13:  return  $gradJ$ 
14: end function

```

As it can be seen, both functions call further two modules:

- *ApplyForwardModel.c* for the application of the model M^k to the vector x through an iterative approach, described in the following pseudocode:

```

1: function applytheforwardmodel(A, B, x, n_x, n_y, h_x, h_y, k)
2:   n ← n_x * n_y
3:   h ← reshape(x(1 : n), n_x, n_y)
4:   u ← reshape(x(n + 1 : 2 * n), n_x, n_y)
5:   v ← reshape(x(2 * n + 1 : 3 * n), n_x, n_y)
6:   U1 ← h
7:   U2 ← h * u
8:   U3 ← h * v
9:   full_overlap_regions(U1)
10:  full_overlap_regions(U2)
11:  full_overlap_regions(U3)
12:
13:  U1dx ← (U1(2 : n_x + 1, :) - U1(1 : n_x, :)) ./ h_x
14:  U2dx ← (U2(2 : n_x + 1, :) - U2(1 : n_x, :)) ./ h_x
15:  U3dx ← (U3(2 : n_x + 1, :) - U3(1 : n_x, :)) ./ h_x
16:
17:  U1dy ← (U1(:, 2 : n_y + 1) - U1(:, 1 : n_y)) ./ h_y
18:  U2dy ← (U2(:, 2 : n_y + 1) - U2(:, 1 : n_y)) ./ h_y
19:  U3dy ← (U3(:, 2 : n_y + 1) - U3(:, 1 : n_y)) ./ h_y
20:
21:  Z1 ← (A(:, :, 1) * U1dx + A(:, :, 2) * U2dx + A(:, :, 3) * U3dx) +
22:        (B(:, :, 1) * U1dy + B(:, :, 2) * U2dy + B(:, :, 3) * U3dy)
23:  Z2 ← (A(:, :, 4) * U1dx + A(:, :, 5) * U2dx + A(:, :, 6) * U3dx) +
24:        (B(:, :, 4) * U1dy + B(:, :, 5) * U2dy + B(:, :, 6) * U3dy)
25:  Z3 ← (A(:, :, 7) * U1dx + A(:, :, 8) * U2dx + A(:, :, 9) * U3dx) +
26:        (B(:, :, 7) * U1dy + B(:, :, 8) * U2dy + B(:, :, 9) * U3dy)
27:  full_overlap_regions(Z1)
28:  full_overlap_regions(Z2)
29:  full_overlap_regions(Z3)
30:  for t = 2 to k do
31:
32:    U1dx ← (Z1(2 : n_x + 1, :) - Z1(1 : n_x, :)) ./ h_x
33:    U2dx ← (Z2(2 : n_x + 1, :) - Z2(1 : n_x, :)) ./ h_x
34:    U3dx ← (Z3(2 : n_x + 1, :) - Z3(1 : n_x, :)) ./ h_x
35:
36:    U1dy ← (Z1(:, 2 : n_y + 1) - Z1(:, 1 : n_y)) ./ h_y
37:    U2dy ← (Z2(:, 2 : n_y + 1) - Z2(:, 1 : n_y)) ./ h_y
38:    U3dy ← (Z3(:, 2 : n_y + 1) - Z3(:, 1 : n_y)) ./ h_y
39:    Z1 ← (A(:, :, 1) * U1dx + A(:, :, 2) * U2dx + A(:, :, 3) * U3dx) +
40:          (B(:, :, 1) * U1dy + B(:, :, 2) * U2dy + B(:, :, 3) * U3dy)
41:    Z2 ← (A(:, :, 4) * U1dx + A(:, :, 5) * U2dx + A(:, :, 6) * U3dx) +
42:          (B(:, :, 4) * U1dy + B(:, :, 5) * U2dy + B(:, :, 6) * U3dy)
43:    Z3 ← (A(:, :, 7) * U1dx + A(:, :, 8) * U2dx + A(:, :, 9) * U3dx) +
44:          (B(:, :, 7) * U1dy + B(:, :, 8) * U2dy + B(:, :, 9) * U3dy)
45:    full_overlap_regions(Z1)
46:    full_overlap_regions(Z2)
47:    full_overlap_regions(Z3)
48:  end for
49:  U1 ← U1 - Z1
50:  U2 ← U2 - Z2
51:  U3 ← U3 - Z3
52:  h_new ← U1
53:  u_new ← U2 ./ U1
54:  v_new ← U3 ./ U1
55:  y(1 : n) ← reshape(h_new, n, 1)
56:  y(n + 1 : 2 * n) ← reshape(u_new, n, 1)
57:  y(2 * n + 1 : 3 * n) ← reshape(v_new, n, 1)
58:  return y

```

- *ApplyBackwardModel.c* : for the application of the model M^{T^k} to the vector x through an iterative approach, described in the following pseudocode:

```

1: function applythebackwardmodel( $A, B, x, n_x, n_y, h_x, h_y, k$ )
2:    $n \leftarrow n_x * n_y$ 
3:    $h \leftarrow \text{reshape}(x(1:n), n_x, n_y)$ 
4:    $u \leftarrow \text{reshape}(x(n+1:2*n), n_x, n_y)$ 
5:    $v \leftarrow \text{reshape}(x(2*n+1:3*n), n_x, n_y)$ 
6:    $U1 \leftarrow h$ 
7:    $U2 \leftarrow h * u$ 
8:    $U3 \leftarrow h * v$ 
9:    $\text{full\_overlap\_regions}(U1)$ 
10:   $\text{full\_overlap\_regions}(U2)$ 
11:   $\text{full\_overlap\_regions}(U3)$ 
12:
13:   $U1dx \leftarrow (U1(2:n_x+1,:) - U1(1:n_x,:))./h_x$ 
14:   $U2dx \leftarrow (U2(2:n_x+1,:) - U2(1:n_x,:))./h_x$ 
15:   $U3dx \leftarrow (U3(2:n_x+1,:) - U3(1:n_x,:))./h_x$ 
16:
17:   $U1dy \leftarrow (U1(:,2:n_y+1) - U1(:,1:n_y))./h_y$ 
18:   $U2dy \leftarrow (U2(:,2:n_y+1) - U2(:,1:n_y))./h_y$ 
19:   $U3dy \leftarrow (U3(:,2:n_y+1) - U3(:,1:n_y))./h_y$ 
20:
21:   $Z1 \leftarrow -(A(:,1) * U1dx + A(:,2) * U2dx + A(:,3) * U3dx) -$ 
22:     $(B(:,1) * U1dy + B(:,2) * U2dy + B(:,3) * U3dy)$ 
23:   $Z2 \leftarrow -(A(:,4) * U1dx + A(:,5) * U2dx + A(:,6) * U3dx) -$ 
24:     $(B(:,4) * U1dy + B(:,5) * U2dy + B(:,6) * U3dy)$ 
25:   $Z3 \leftarrow -(A(:,7) * U1dx + A(:,8) * U2dx + A(:,9) * U3dx) -$ 
26:     $(B(:,7) * U1dy + B(:,8) * U2dy + B(:,9) * U3dy)$ 
27:   $\text{full\_overlap\_regions}(Z1)$ 
28:   $\text{full\_overlap\_regions}(Z2)$ 
29:   $\text{full\_overlap\_regions}(Z3)$ 
30:  for  $t = 2$  to  $k$  do
31:
32:     $U1dx \leftarrow (Z1(2:n_x+1,:) - Z1(1:n_x,:))./h_x$ 
33:     $U2dx \leftarrow (Z2(2:n_x+1,:) - Z2(1:n_x,:))./h_x$ 
34:     $U3dx \leftarrow (Z3(2:n_x+1,:) - Z3(1:n_x,:))./h_x$ 
35:
36:     $U1dy \leftarrow (Z1(:,2:n_y+1) - Z1(:,1:n_y))./h_y$ 
37:     $U2dy \leftarrow (Z2(:,2:n_y+1) - Z2(:,1:n_y))./h_y$ 
38:     $U3dy \leftarrow (Z3(:,2:n_y+1) - Z3(:,1:n_y))./h_y$ 
39:     $Z1 \leftarrow -(A(:,1) * U1dx + A(:,2) * U2dx + A(:,3) * U3dx) -$ 
40:       $(B(:,1) * U1dy + B(:,2) * U2dy + B(:,3) * U3dy)$ 
41:     $Z2 \leftarrow -(A(:,4) * U1dx + A(:,5) * U2dx + A(:,6) * U3dx) -$ 
42:       $(B(:,4) * U1dy + B(:,5) * U2dy + B(:,6) * U3dy)$ 
43:     $Z3 \leftarrow -(A(:,7) * U1dx + A(:,8) * U2dx + A(:,9) * U3dx) -$ 
44:       $(B(:,7) * U1dy + B(:,8) * U2dy + B(:,9) * U3dy)$ 
45:     $\text{full\_overlap\_regions}(Z1)$ 
46:     $\text{full\_overlap\_regions}(Z2)$ 
47:     $\text{full\_overlap\_regions}(Z3)$ 
48:  end for
49:   $U1 \leftarrow U1 - Z1$ 
50:   $U2 \leftarrow U2 - Z2$ 
51:   $U3 \leftarrow U3 - Z3$ 
52:   $h_{\text{new}} \leftarrow U1$ 
53:   $u_{\text{new}} \leftarrow U2./U1$ 
54:   $v_{\text{new}} \leftarrow U3./U1$ 
55:   $y(1:n) \leftarrow \text{reshape}(h_{\text{new}}, n, 1)$ 
56:   $y(n+1:2*n) \leftarrow \text{reshape}(u_{\text{new}}, n, 1)$ 
57:   $y(2*n+1:3*n) \leftarrow \text{reshape}(v_{\text{new}}, n, 1)$ 
58:  return  $y$ 

```

The functional J , returned from the call to `ComputeJ.c` function, is numerically minimized using the quasi-Newton method L-BFGS in order to measure the difference between the system state, predicted by the model, and the observed state.

In numerical optimization, the limited-memory Broyden–Fletcher–Goldfarb–Shanno (L-BFGS) algorithm is an iterative method, described by a collection of Fortran 77 routines for solving bound-constrained nonlinear optimization problems.

The L-BFGS method approximates Newton's method, a class of hill-climbing optimization techniques that seeks a stationary point of a function.

In the case of our application iterations of minimizer stop when the absolute value of the gradient of the function $\nabla J(v)$ becomes relatively smaller than its initial value.

The termination condition is evaluated by specifying a tolerance parameter, named EPS, under which the process exits from the loop minimization

In the last part of the application the accuracy of the model is estimated by comparing the initial data structures with structures returned from the minimization process.

Paragraph 2.2: Profiling DD-4DVAR sequential

Developed and performed a first correct sequential version of the application under consideration, the following step was to carry out a dynamic profiling in order to estimate the program in terms of spatial and temporal complexity.

The aim was therefore to have a clear idea on the management of memory allocated by the program and on the most computationally expensive functions than the total program execution time with a view to its subsequent optimization in an execution parallel context.

As regards memory management, in this thesis was used Valgring tool. It looks like a framework, written in C language for the GNU / Linux operating systems, which includes within it various types of debug tools.

In our case, the attention was focused on MemCheck tool to identify possible memory

leaks in the code, wrong accesses to memory locations or overlapping pointers.

The second phase of the profiling focused on the study of the temporal aspects of the application and therefore on identifying those portions of code that employ more time in their execution.

The instrument used in this case is **Gprof**, a performance analysis tool for Unix applications.

Two forms of output are used in this analysis:

- *Flat Profile*: shows the amount of time that each function constituting the program employs and the total number of times that the function is called
- *Call Graph*: shows, for each function, which functions called it, which other functions it called, and how many times. There is also an estimate of how much time was spent in the subroutines of each function.

The result is the following:

Call graph

index	%time	self	children	called	name
[1]	100.0	15.43	248.08	1/1	main [2]
		15.43	248.08	1	subdomain [1]
		141.28	0.25	3/3	computeGradJ [3]
		106.48	0.07	3/3	Jfunction [4]
		0.01	0.00	1/1	buildmodels [7]
		0.00	0.00	14/14	get_time [8]
		0.00	0.00	6/6	findHighest [9]
		0.00	0.00	3/3	s_lbfgs [13]

[2]	100.0	0.00	263.51		<spontaneous>
		15.43	248.08	1/1	subdomain [1]

[3]	53.7	141.28	0.25	3/3	subdomain [1]
		141.28	0.25	3	computeGradJ [3]
		0.18	0.00	33/33	applybackwardmodel [5]
		0.07	0.00	33/66	applyforwardmodel [6]

[4]	40.4	106.48	0.07	3/3	subdomain [1]
		106.48	0.07	3	Jfunction [4]
		0.07	0.00	33/66	applyforwardmodel [6]

[5]	0.1	0.18	0.00	33/33	computeGradJ [3]
		0.18	0.00	33	applybackwardmodel [5]

[6]	0.0	0.07	0.00	33/66	Jfunction [4]
		0.07	0.00	33/66	computeGradJ [3]
		0.13	0.00	66	applyforwardmodel [6]

[7]	0.0	0.01	0.00	1/1	subdomain [1] buildmodels [7]
[8]	0.0	0.00	0.00	14/14 14	subdomain [1] get_time [8]
[9]	0.0	0.00	0.00	6/6 6	subdomain [1] findHighest [9]
[10]	0.0	0.00	0.00	1/4 3/4 4	s_lbfgs [13] s_mcsrch [14] MAX [10]
[11]	0.0	0.00	0.00	4/4 4	s_mcsrch [14] MIN [11]
[12]	0.0	0.00	0.00	3/3 3	mcstep [16] mymax [12]
[13]	0.0	0.00	0.00	3/3 3 3/3 2/2 0.00 0.00 1/4	subdomain [1] s_lbfgs [13] s_mcsrch [14] s_lb1 [15] MAX [10]
[14]	0.0	0.00	0.00	3/3 3 0.00 0.00 4/4 0.00 0.00 3/4 0.00 0.00 1/1	s_lbfgs [13] s_mcsrch [14] MIN [11] MAX [10] mcstep [16]
[15]	0.0	0.00	0.00	2/2 2	s_lbfgs [13] s_lb1 [15]
[16]	0.0	0.00	0.00	1/1 1 3/3 1/1	s_mcsrch [14] mcstep [16] mymax [12] mymin [17]
[17]	0.0	0.00	0.00	1/1 1	mcstep [16] mymin [17]

index A unique number given to each element of the table.
Index numbers are sorted numerically.
The index number is printed next to every function name so
it is easier to look up where the function is in the table.

%time This is the percentage of the 'total' time that was spent
in this function and its children. Note that due to
different viewpoints, functions excluded by options, etc,
these numbers will NOT add up to 100%.

self This is the total amount of time spent in this function.

children This is the total amount of time propagated into this
function by its children.

called This is the number of times the function was called.
If the function called itself recursively, the number
only includes non-recursive calls, and is followed by
a '+' and the number of recursive calls.

name The name of the current function.|

Flat profile:

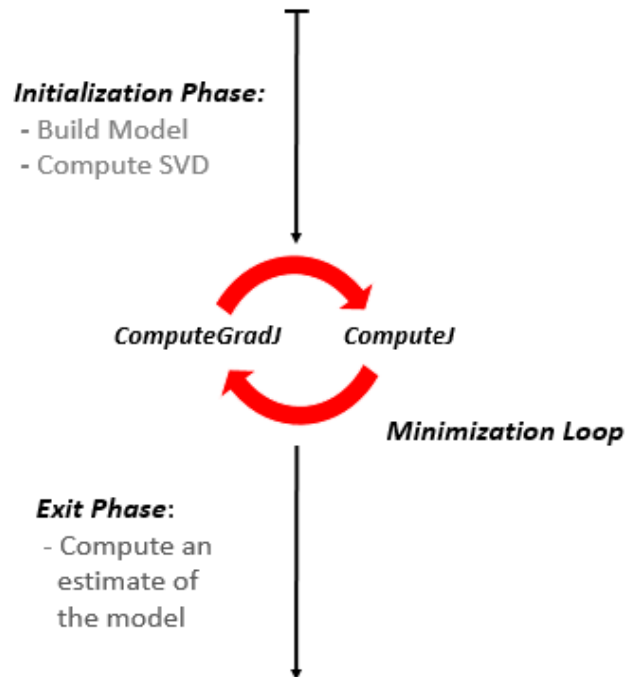
Each sample counts as 0.01 seconds.

% time	cumulative seconds	self seconds	self calls	total s/call	s/call	name
53.62	141.28	141.28	3	47.09	47.18	computeGradJ
40.41	247.76	106.48	3	35.49	35.51	Jfunction
5.86	263.19	15.43	1	15.43	263.51	subdomain
0.07	263.37	0.18	33	0.01	0.01	applybackwardmodel
0.05	263.50	0.13	66	0.00	0.00	applyforwardmodel
0.00	263.51	0.01	1	0.01	0.01	buildmodels
0.00	263.51	0.00	14	0.00	0.00	get time
0.00	263.51	0.00	6	0.00	0.00	findHighest
0.00	263.51	0.00	4	0.00	0.00	MAX
0.00	263.51	0.00	4	0.00	0.00	MIN
0.00	263.51	0.00	3	0.00	0.00	mymax
0.00	263.51	0.00	3	0.00	0.00	s_lbfgs
0.00	263.51	0.00	3	0.00	0.00	s_mcsrch
0.00	263.51	0.00	2	0.00	0.00	s_lb1
0.00	263.51	0.00	1	0.00	0.00	mcstep
0.00	263.51	0.00	1	0.00	0.00	mymin

- **% time** : the percentage of the total running time of the program used by this function.
- **cumulative seconds** : a running sum of the number of seconds accounted for by this function and those listed above it.
- **self seconds** : the number of seconds accounted for by this function alone.
- **calls** : the number of times this function was invoked, if this function is profiled, else blank.
- **self ms/call** : the average number of milliseconds spent in this function per call, if this function is profiled, else blank.
- **total ms/call** : the average number of milliseconds spent in this function and its descendents per call, if this function is profiled, else blank.
- **name** : the name of the function.

Analyzed in detail the results of the profiling, is possible to see that the program spends about 94% of the total time in performing *ComputeJ.c* and *ComputeGradJ.c* functions and their submodules, which characterize the various iterations of the minimization process.

The program structure can therefore be represented as follows:



In addressing the optimization process, the following aspects were evaluated:

- The initialization phase as well as the final stage, is performed just one time by each process, so as long it can be, it presents an insignificant impact on performance
- The phase of the loop, instead, as is clearly seen from the results of the profiling, turns out to be the crucial point on which focus on.

In fact, every second or in general every portion of time gained in one iteration, is multiplied many times as the loop is executed.

Chapter 3: Parallel version DD4DVAR

Starting from the results of the sequential model described in the previous chapter, in the second part of this work of thesis it was developed a parallel version of the application in C language using the programming style MPI (Message Passing Interface).

The context in which measurements are made is the Project Mont Blanc in the BSC seat (Barcelona Supercomputing Center) which will be described in detail in the next paragraph.

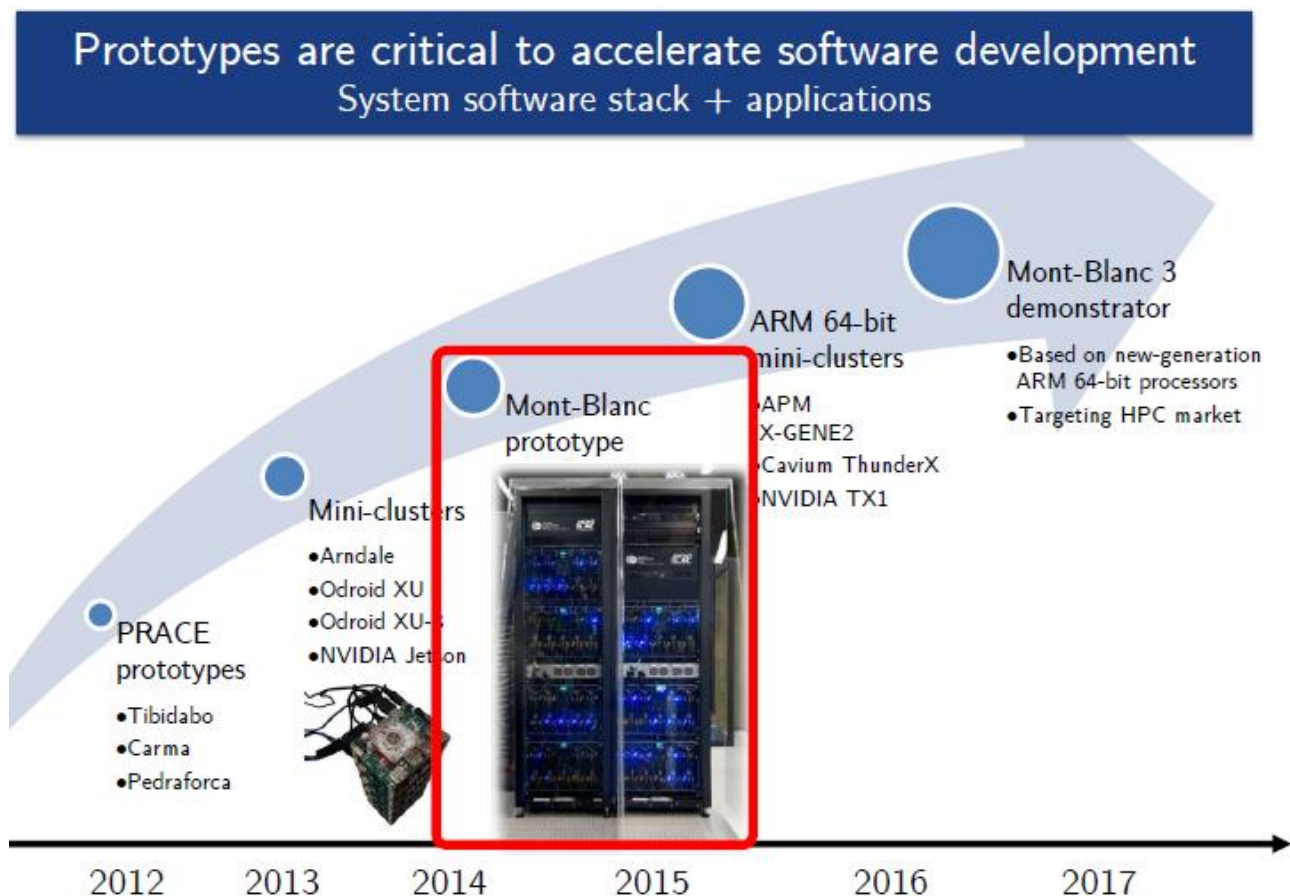
The choice to run this application in this specific computer center is dictated by the opportunity to extract, in addition to traditional measures of performance of the HPC world, also significant results in an innovative field such as the one of the energy.

Paragraph 3.1: Mont-Blanc Project

Energy efficiency is already a primary concern for the design of any computer system and it is unanimously recognized that future Exascale systems will be strongly constrained by their power consumption. This is why the Mont-Blanc project has set itself the following objective: to design a new type of computer architecture capable of setting future global High Performance Computing (HPC) standards that will deliver Exascale performance while using 15 to 30 times less energy.

This project is coordinated by the Barcelona Supercomputing Center (BSC) and aims at the following objectives:

1. To complement the effort on the Mont-Blanc system software stack, with emphasis on programmer tools and ARM 64-bit support.
2. To produce a first definition of the Mont-Blanc Exascale architecture, exploring different alternatives for the compute node (from low-power mobile sockets to special-purpose high-end ARM chips), and its implications on the rest of the system
3. To track the evolution of ARM-based systems.



A more detailed description of the Mont-Blanc system and of the individual components is the following:

Exynos 5 compute card

2 x Cortex-A15 @ 1.7GHz
 1 x Mali T604 GPU
 6.8 + 25.5 GFLOPS
 15 Watts
 2.1 GFLOPS/W

**Carrier blade**

15 x Compute cards
 485 GFLOPS
 1 GbE to 10 GbE
 300 Watts
 1.6 GFLOPS/W

**Blade chassis 7U**

9 x Carrier blade
 135 x Compute cards
 4.3 TFLOPS
 2.7 kWatts
 1.6 GFLOPS/W

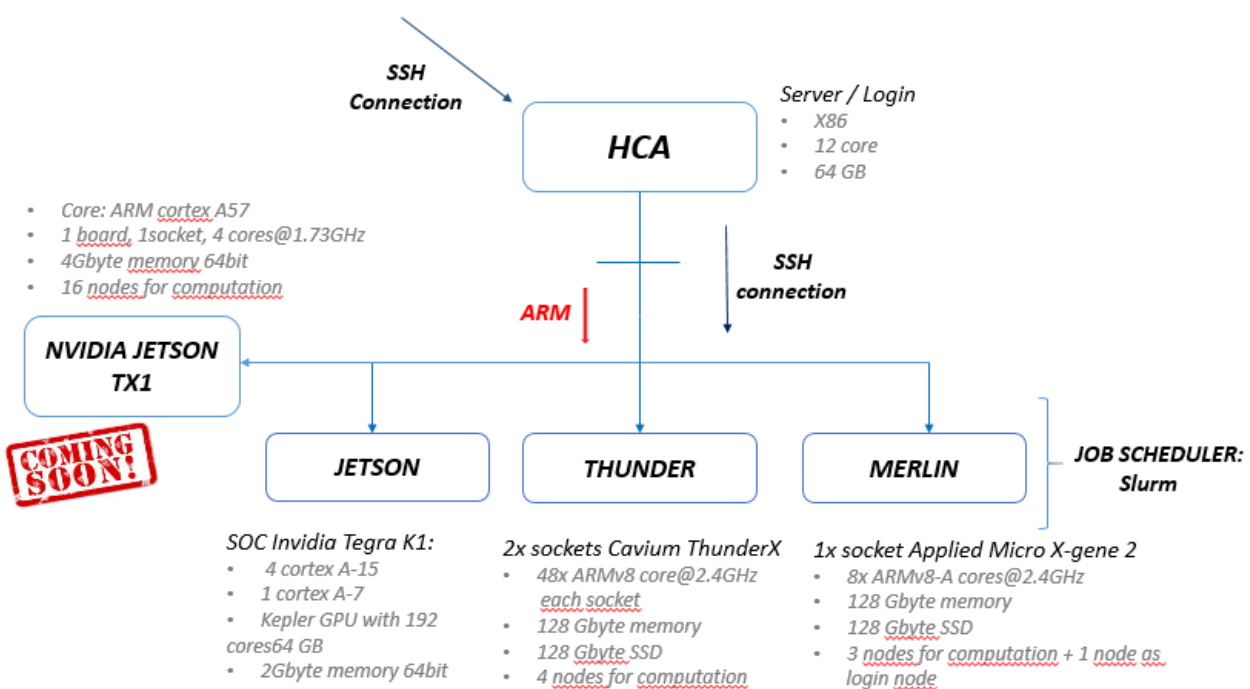
**Rack**

8 BullX chassis
 72 Compute blades
 1080 Compute cards
 2160 CPUs
 1080 GPUs
 4.3 TB of DRAM
 17.2 TB of Flash

35 TFLOPS
24 kWatt

Paragraph 3.1.1: Environment of application's development

The application was compiled and developed in the following contexts:



In this paragraph is described the design and the implementation of a first parallel version of the application under consideration.

The independence in the evaluation of the final results allowed a work decomposition across multiple processors, making the problem well suited to parallel implementation. To implement a Data Assimilation algorithm in oceanography, it is natural to think about Domain Decomposition (DD) approach: each sub domain of the decomposition could be associated to a geographical area which could have its own observational and computational resources.

The DD approach agrees with the physical characteristics of the ocean model: the 3D domain is laid out on local processor memories following a 2D horizontal topological splitting. Further, each sub domain computes its own surface and bottom boundary conditions and it has a side wall overlapping interface which defines the lateral boundary conditions for computations in the inner sub domain. The overlapping area consists of the two rows at each edge of the sub domain. After a computation, a communication phase starts: each processor sends to its neighboring processors the updated values of the points corresponding to the interior overlapping area of its neighboring sub domain (i.e. the innermost of the two overlapping rows). The communication is done through message passing.

In DD-OceanVar we use this DD approach and the same overlapping area for the exchange of information on boundary conditions.

The reasons behind this choice are mainly two:

- Decompose the original problem, having an initial dataset of three $N \times N$ matrices, on an arbitrary number of processes with a problem size of smaller dimension.
- Build a topology in which we can imagine disposed processes, in order to optimize the MPI communications. In essence, a "virtual" geometry in which the processors are organized according to a space division of the North, South, East, West.

The solution adopted in this context was to use a programming approach SIMD with distributed memory and consequently a style of programming MPI (Message Passing Interface) using the library installed on the remote cluster OpenMPI.

The main phases, concerned by the parallel analysis process, are essentially the initialization phase and the most onerous computationally functions identified in the profiling phase, present in the minimization loop:

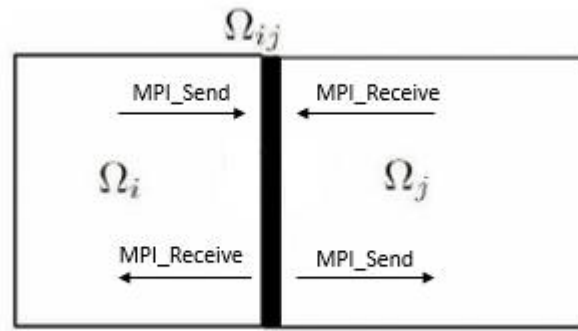
- In the initialization phase, were generated the size vector and the displacement vector, that is the memory space contiguous concerning the subvector to be sent to each process and the displacements needed to manage the pointers to the subvectors. At this point there has been a distribution of subvectors to each processor belonging to the working group.

The MPI library provides collective communication functions such as `Scatterv()` that involves all the processes of the communicator, in order to optimize the communication times.

- In the minimization phase, parallelization involved the following modules:
 - *ComputeJ.c*: For the purposes of the calculation of the functional $J(v)$ each process, at the end of its elaborations, adds a final contribution, calculated using the knowledge of a value belonging to its "neighbour" in the MPI environment to enforce coupling between the domains.
 - *Applyforwardmodel.c* and *Applybackwardmodel.c*: recall, in turn, a new module named *filloverlap.c* dedicated to the management of the "border" segment that the various processes share with each other.
 - *filloverlap.c*: the processes that fall under the category "West" with the "East" ones and those classified as "North" with the "South" ones, through `MPI_Isend()` and `MPI_Irecv()`, exchange the values belonging to the "border" segment in common.

Let us consider the following overlapping decomposition of the physical domain Ω :

$$\Omega = \bigcup_{i=1}^N \Omega_i$$



such that $\Omega_i \cap \Omega_j = \Omega_{ij} \neq 0$ if Ω_i and Ω_j are adjacent.

Paragraph 3.3: Results

In order to estimate the goodness of parallelization used, have been carried out a series of measurements that have affected before the HCA Server, having the characteristics described previously, and with Intel technology, and then the Thunder cluster with ARM technology.

The evaluation of the parallel algorithm performances, in its two functions of interest, was made by comparing the processing times of the sequential version with those of the parallel version, calculating the speedup and the efficiency by changing the number of p processors.

Said:

- $T(s)$ the execution time of the best sequential algorithm
- $T(p)$ the execution time of the corresponding parallel algorithm

We fix:

- Speedup: $\mathbf{Sp} = \mathbf{T_s} / \mathbf{T_p}$
- Efficiency: $\mathbf{Ep} = \mathbf{Sp} / \mathbf{p}$

In order to obtain an assessment as close as possible to actual values of the proposed algorithms, all executions are carried out:

- Changing the number of processors (parallel event) in the set $p \in \{1,2,4,8\}$ in the case of analysis on HCA and in the set $p \in \{1,2,4,8,16\}$ in the case of analysis on the Thunder.
- Repeating the same experiment at least ten times and considering the mean value of the results, in order to minimize the possible variance of the data linked to different system loading conditions.

It is important to underline that, according to Amdahl's law, time T_s should be calculated using the best sequential algorithm available for the given problem, but it is often difficult, if not impossible, to determine which is the best usable algorithm.

The choice made in this work consists in confusing $T_s = T_p$ (1) that is, in considering the processing time in the sequential case coincident with the parallel algorithm executed on a single processor.

Although this choice presents potential intrinsic errors (related to the potential additional operations performed by a parallel algorithm than its sequential equivalent), it is extremely advantageous from an implementation point of view, producing at the same time results not too many away from the real case (use of the best sequential algorithm).

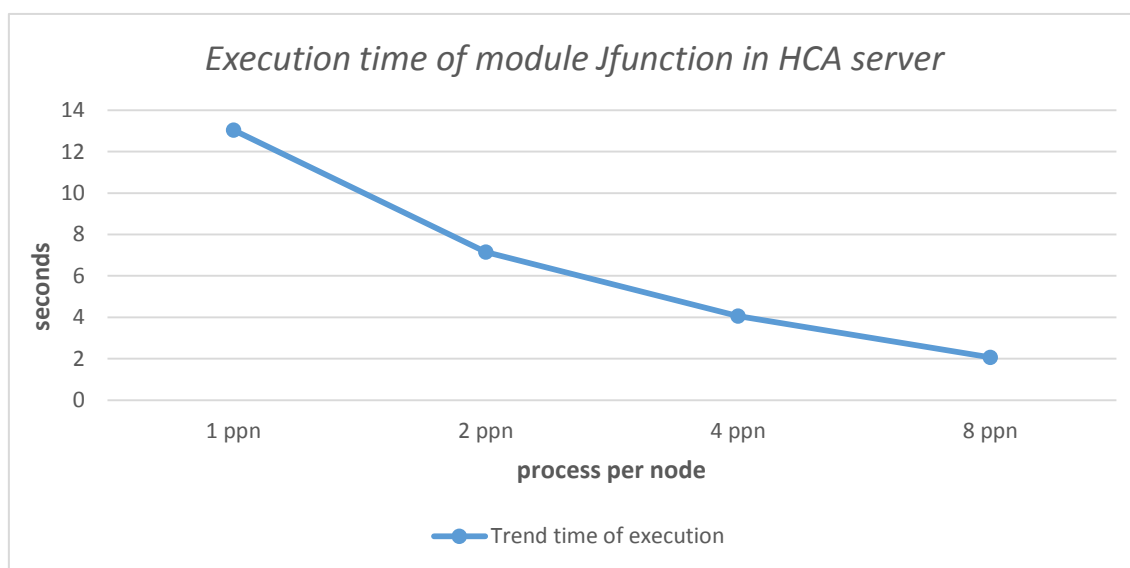
Paragraph 3.3.1: Analysis in HCA

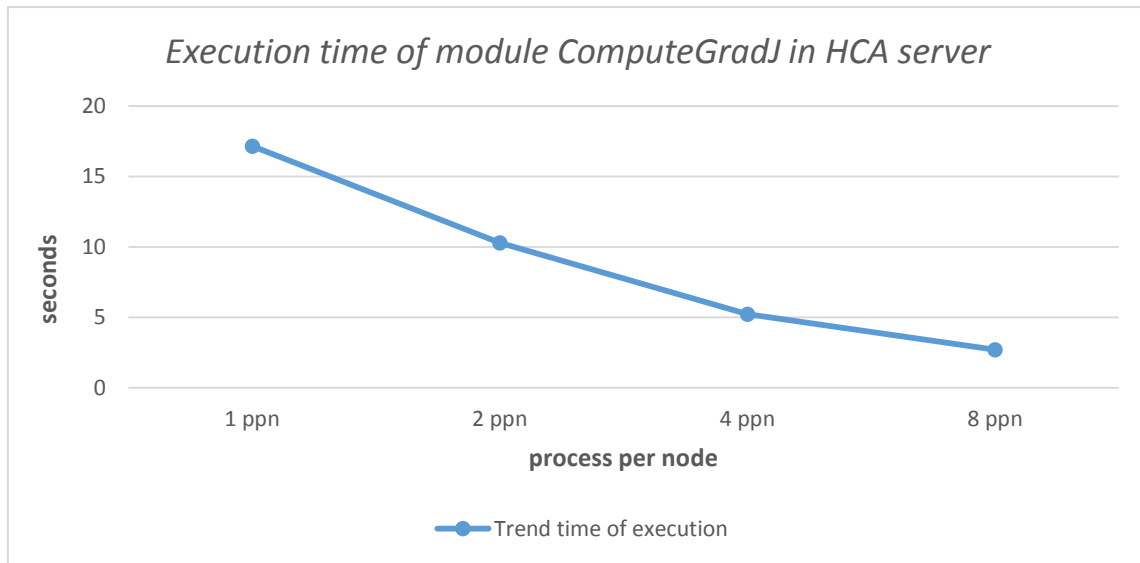
The following tables show the average time, the values of SpeedUp and Efficiency, relating to the two functions under consideration, recorded in various executions by changing the number of processes:

<i>Analisi del modulo Jfunction</i>			
Process per node	Execution Time	SpeedUp	Efficiency
1 ppn	13,0452	1	100
2 ppn	7,1545	1,8323	91
4 ppn	4,0565	3,2158	80
8 ppn	2,0661	6,3139	78

<i>Analisi del modulo ComputeGradJ</i>			
Process per node	Execution Time	SpeedUp	Efficiency
1 ppn	17,1476	1	100
2 ppn	10,285	1,6672	84
4 ppn	5,2123	3,2976	82
8 ppn	2,6911	6,3745	79

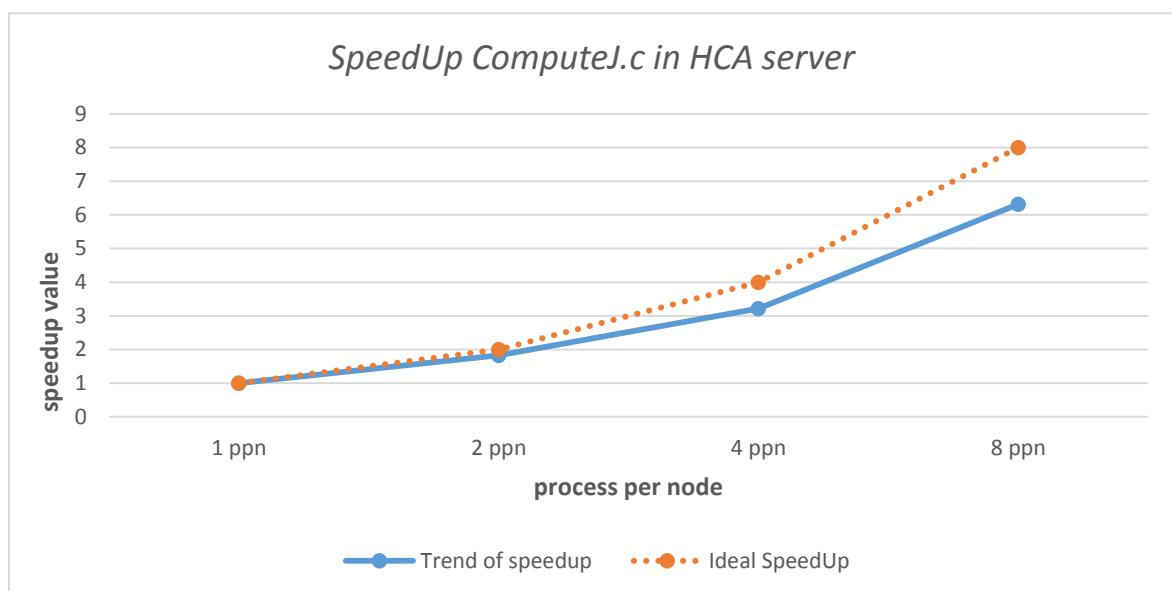
The following figure shows the trends of the execution time by changing the number p of processes per node, in the case of Jfunction and ComputGradJ modules:

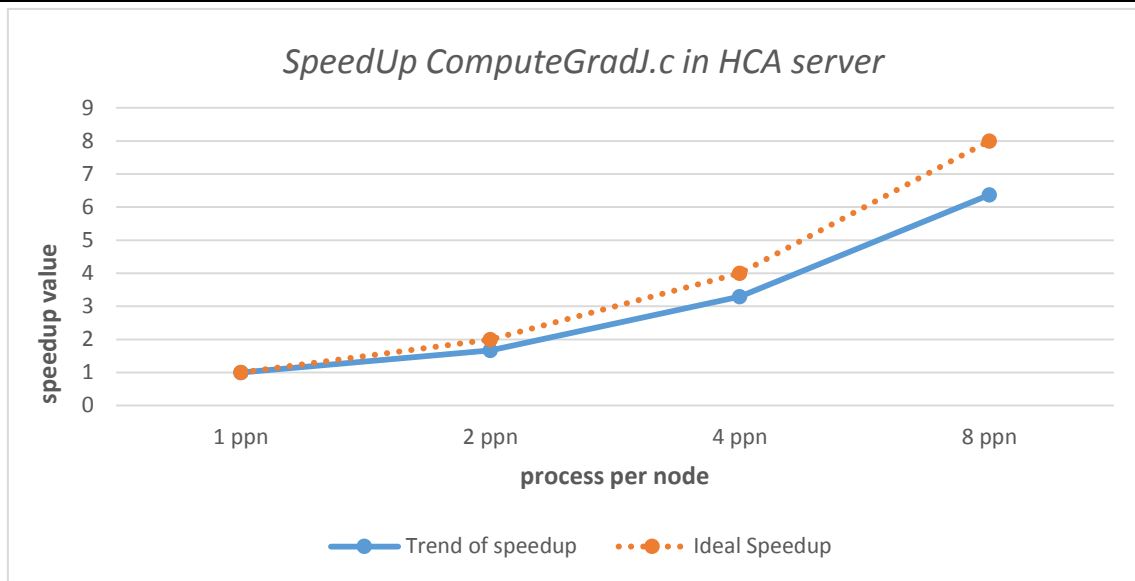




As is possible to see, changing the number of processes per node, the execution time decreases with an almost exponential trend.

The following figure shows the trends of the speedup values by changing the number p of processes per node.

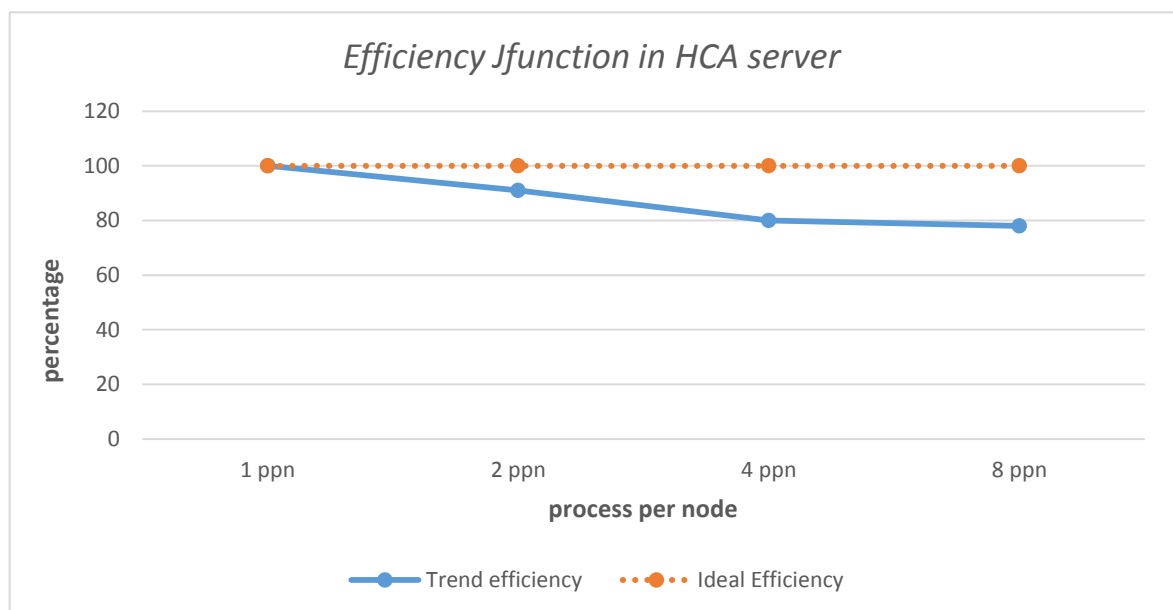


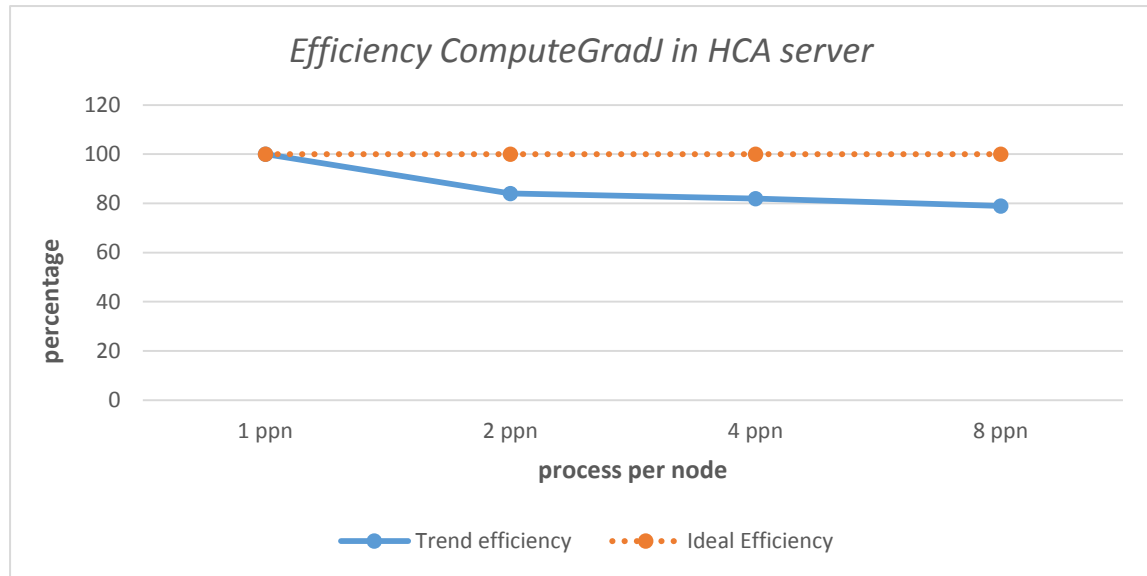


Considering the case of Jfunction module we can observe how in correspondence of the process value per node equal to 2, the speedup presents an acceptable value equal to 1.8323. Value which gradually deteriorates moving away from the ideal speedup coincident with the number of processes, because it is limited by the sequential part of the algorithm.

The same situation is also in the case of ComputeGradJ function.

In the following graph we analyzed the efficiency values in each case of measuring:





As was expected, in both cases of study, the curve highlights how the algorithm shows few efficiency in its implementation for any value of ppn considered, reaching its maximum, that is, a threshold equal to 90% in correspondence of the values of ppn = 2 and the minimum of 63% with ppn = 8.

The reason of this low efficiency is easily identifiable in a constant initial dataset size (the fraction of sequential time is constant) in the face of the increase in the number of processes.

As part of a realistic study of the application, we can imagine how the size of the problem is orders of magnitude far superior to the dataset object of study in this thesis, leading to a more efficient use of the processor.

Paragraph 3.3.2: Analysis in Thunderx cluster

Before starting the discussion on the application's performance in the case of cluster Thunder it is appropriate to make a short digression on some key concepts used in the optimization path.

Thunderx is presented as the first ARM based SoC that scales up to 48 cores with up to 2.5 GHz core frequency but mainly as the first ARM based SoC to be fully cache

coherent across dual sockets using Cavium Coherent Processor Interconnect (CCPI).

In fact, the Thunderx chips support NUMA clustering that allow for two processors to be lashed together to create a shared memory space.

One aspect that has been observed is therefore how to handle NUMA mechanism in the execution of the application.

In effect, when generally an application is executed for example with the command:

```
srun --ntasks=32 ./executable
```

this does not ensure that only one socket is used, this could lead to bad performance.

It is therefore necessary to ensure that only one socket is used if possible, this is guaranteed with the use of *numactl* option that control NUMA policy among processes, or by specifying the mask of the socket used with the option *cpu_bind*.

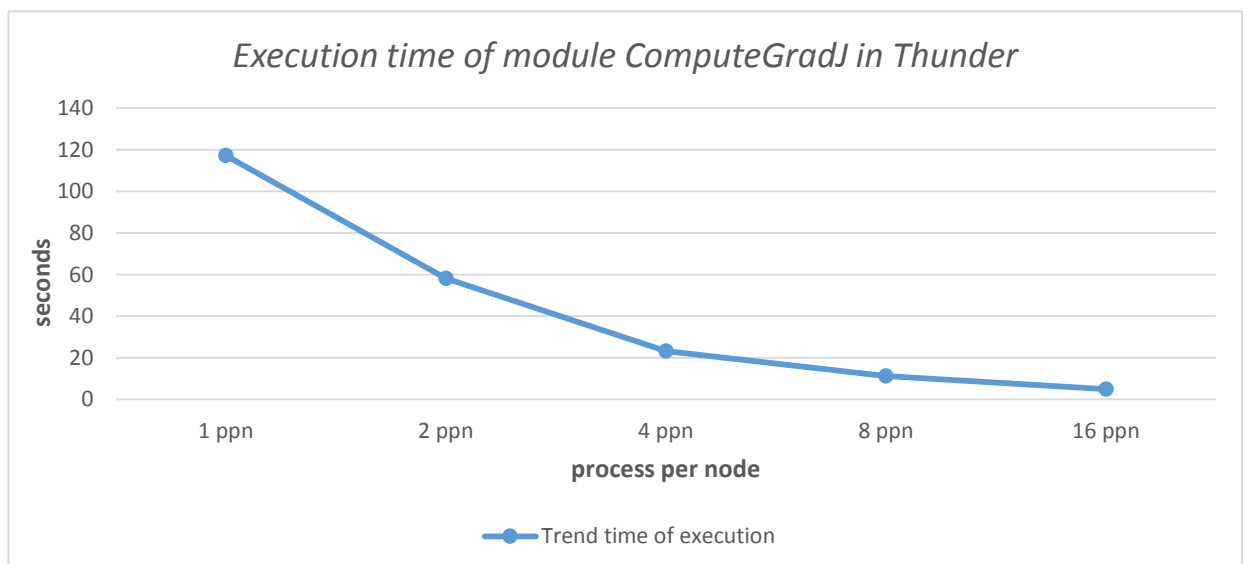
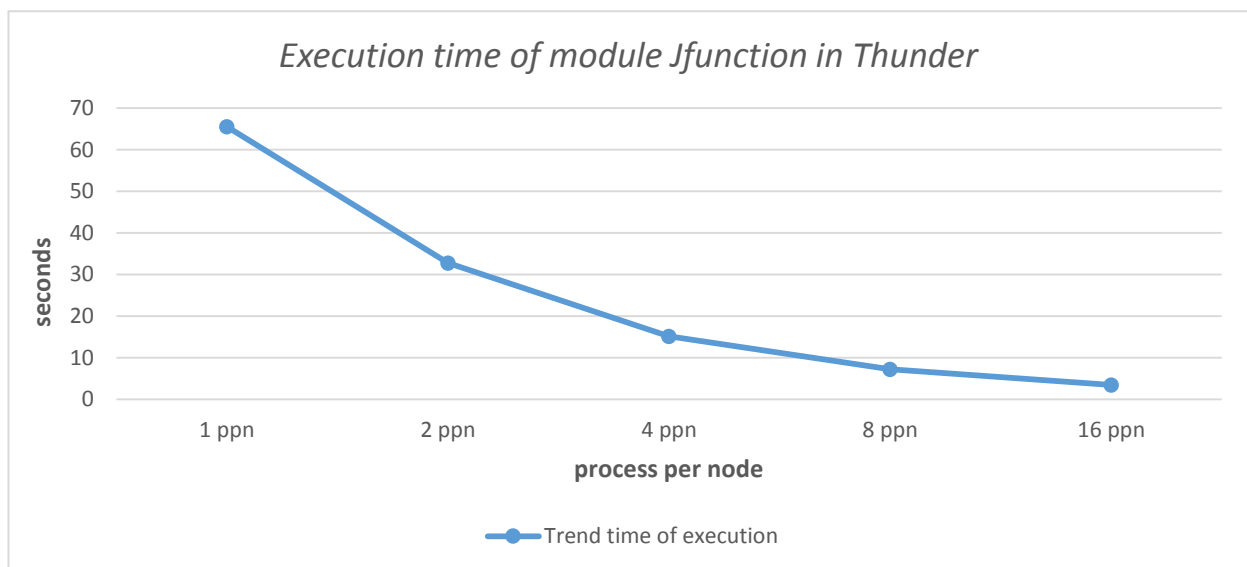
The following tables show the average time recorded in the various executions, the values of SpeedUp and efficiency, by changing the number of processes.

The observations were carried out through the submission of the program to the Job Scheduler SLURM of the platform:

Anàlisi del mòdul Jfunction			
Process per node	Execution Time	SpeedUp	Efficiency
1 ppn	65,4895	1	100
2 ppn	32,7198	2,0015	100,75
4 ppn	15,1514	4,3223	108
8 ppn	7,1697	9,1342	114
16 ppn	3,4112	19,1983	119

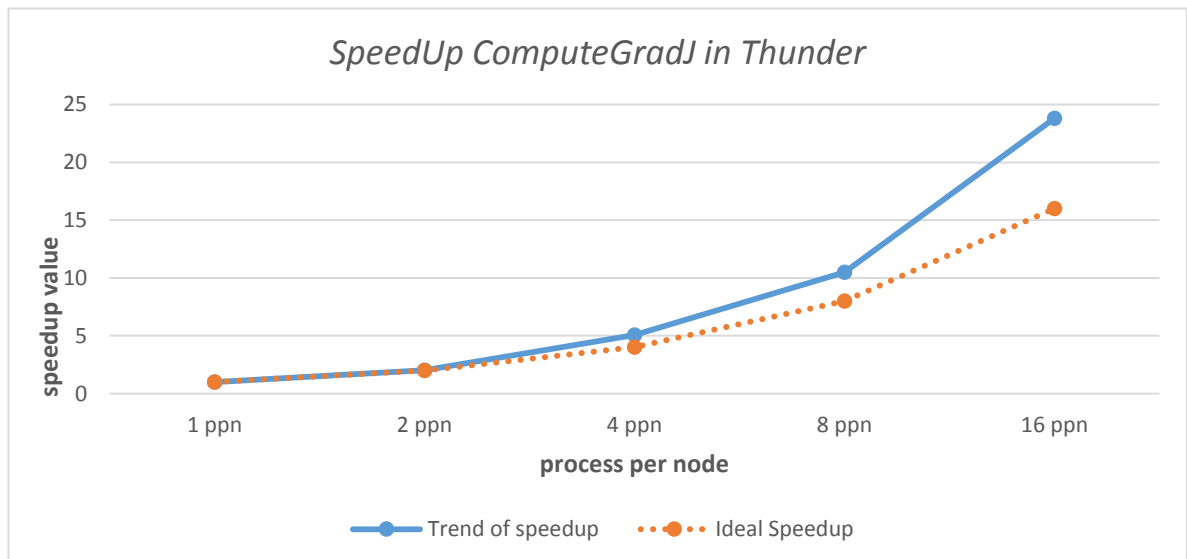
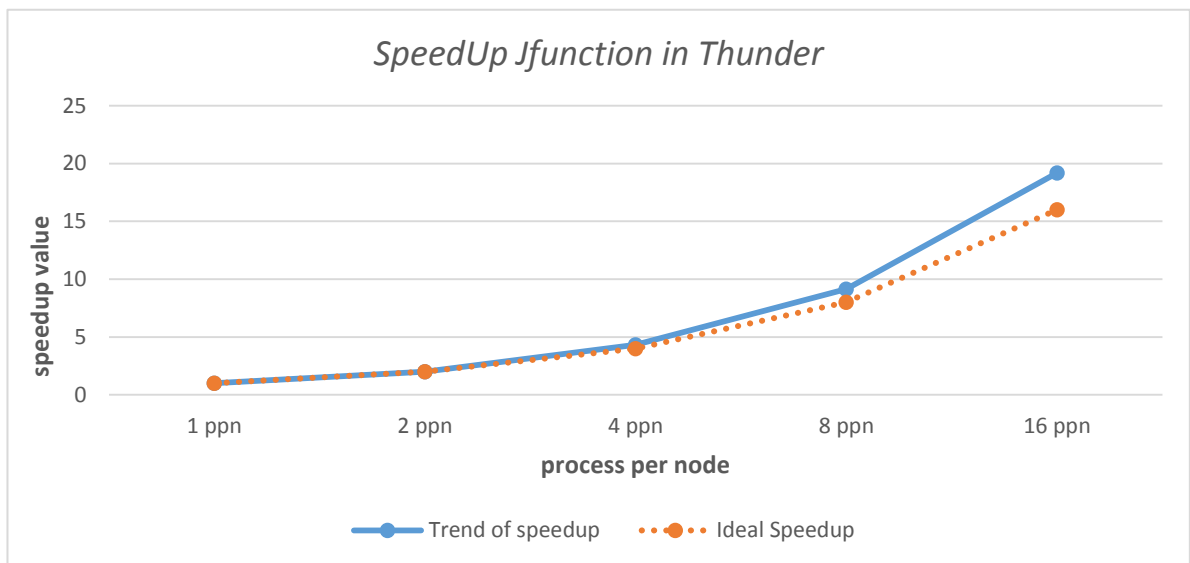
Anàlisi del mòdul ComputeGradJ			
Process per node	Execution Time	SpeedUp	Efficiency
1 ppn	117,1476	1	100
2 ppn	58,1285	2,0197	101
4 ppn	23,1569	5,0588	126
8 ppn	11,1864	10,5001	131
16 ppn	4,9217	23,8022	148

The following figure shows the trends of the execution time by changing the number p of processes per node, in the case of Jfunction and ComputeGradJ module:



As we can see, the execution times, in the case of the running on mini-cluster Thunder, are higher than the observations in HCA.

The reason of this increase can be attributed to a greater number of memory accesses from the program for the taking of data due to a smaller size of the cache.

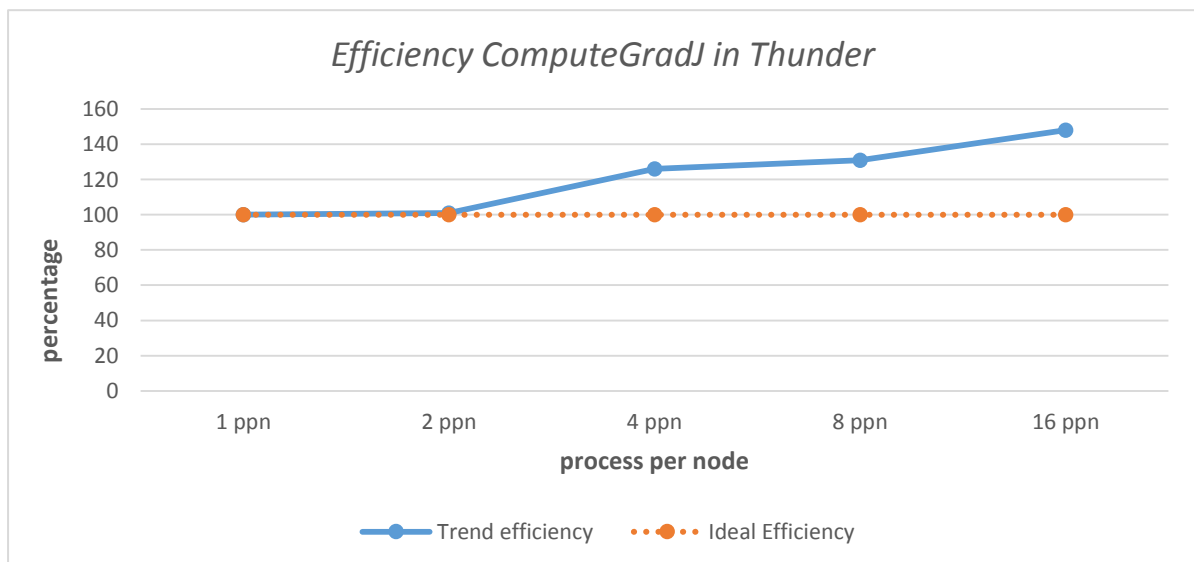
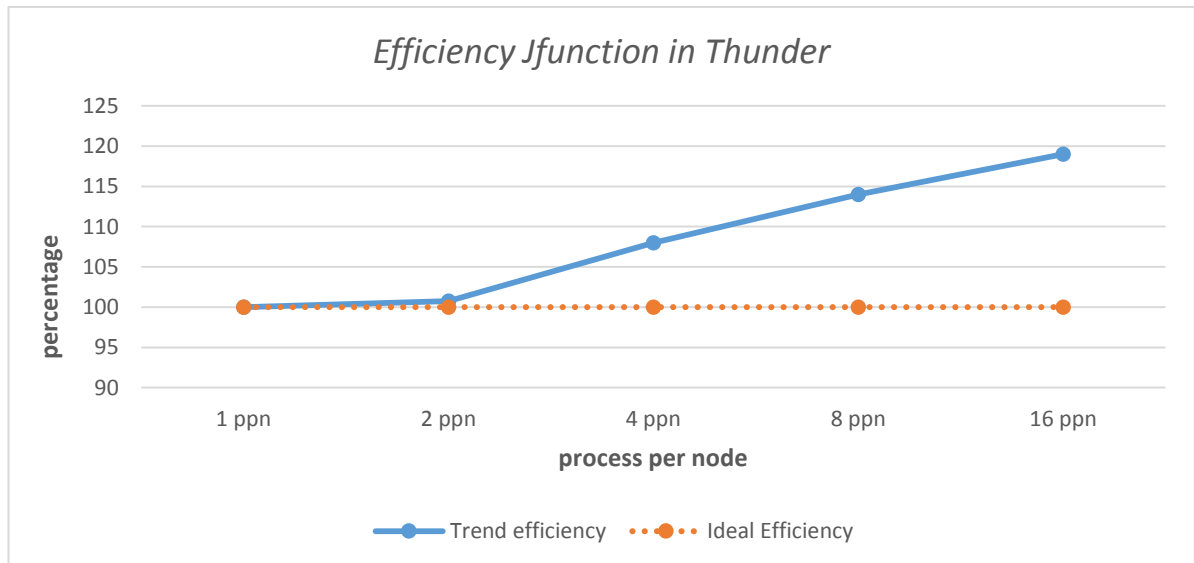


The curves show how, with increasing number p of process per node, the speedup measures take greater values than the ideal speedup.

This trend is labeled as "SuperLinear Speedup" and, in our case, it can be explained by

a more efficient use of resources compared with the sequential case.

One example is the reduction of RAM access time, this happens when the working set of a problem is greater than the cache size when executed sequentially, so the CPUs have to access main memory, which takes hundreds of clock cycles; this situation can fit nicely in each available cache when executed in parallel.



Chapter 4: Power Efficiency

In a world with limited energy resources and a rising demand for more computational power, energy consumption is limiting the scale of computers that can be deployed.

So, to build exascale systems, it is necessary to improve energy efficiency (EE) (Flops/Watt) of current systems.

This chapter presents a review about High Power Consumption of High Performance Computing systems, one of the major hurdles in the path to exascale system, before in general terms and then specifically for the application.

It is organized as follow:

- The section 4.1 introduces the concepts of *power consumption*, *power efficiency* and rank lists.
- The section 4.2 shows the problem to build exascale system
- The section 4.3 shows how was developed the study of the power efficiency in the context of this thesis.

Paragraph 4.1: Supercomputer performances, Power Consumption and rank lists

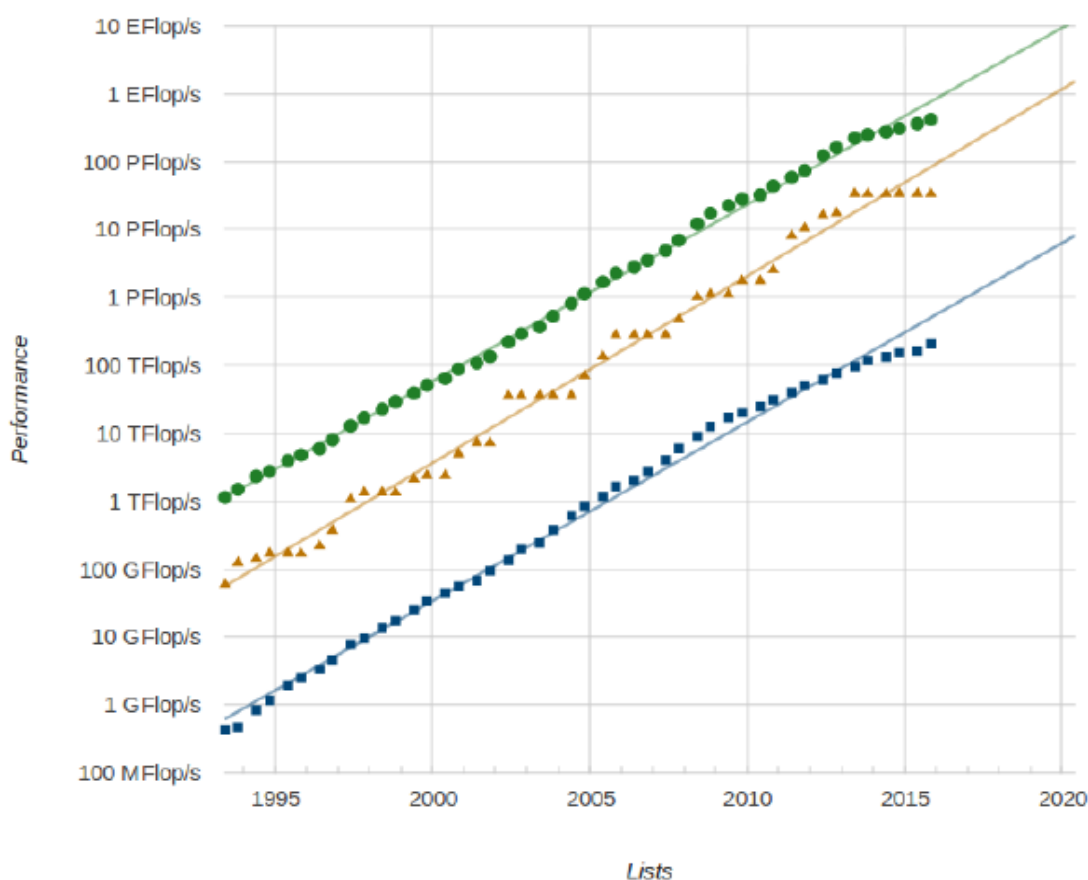
In performance computing context,

- **Flops** (an acronym for **floating-point operations per second**) is a measure of computer performance, useful in fields of scientific calculations that make heavy use of floating-point calculations.

- **Flops/watt** is a measure of the energy efficiency of a particular computer architecture or computer hardware. Literally, it measures the rate of computation that can be delivered by a computer for every watt of power consumed.

Since 1993, the Top500 list offers a ranking of the 500 fastest supercomputers. Running the Linpack benchmark, the supercomputers are compared and ranked according to their performance.

How is possible to see in the following graphic of this community:



The projections for exaflop-scale computing systems, assuming (non-realistic) linear Scale, suggest how is necessary a system ~1000x larger of a current system in order to reach one EFLOP.

This means ~1 GWatts for operating a supercomputer with this features, which is the energy generated by Vandellòs Nuclear Power Plant in Catalunya.

Therefore, with today's technology, exascale system would consume over a gigawatt of power, making it economically and ecologically impracticable.

Given the current energy consumption of exaflop-scale systems, and considering the limited supply of energy, interest in this area has increased and the analysis of power efficiency has also been used to evaluate systems.

The green computing area is becoming increasingly important in a world with limited energy resources. With this concern, the Green500 list was created to provide a ranking of the most energy efficient supercomputers.

So, the Top500 List offers a ranking of the 500 fastest supercomputers (Flops) and the Green500 List, ranks the top 500 supercomputers in the world by energy efficiency (Flops/Watt).

The performance increase of HPC systems has been achieved with the increase of processors/cores and, recently, by adding accelerators such as graphic processor units (GPU) which guarantees a better efficiency energy.

Paragraph 4.2: Research in Exascale Systems

Today, to build exascale systems, it is necessary to analyze the problem of performance from a different point of view.

As we know, the throughput of a system, ie the number of floating point operations per second, it can be estimated in this way

$$Throughput = \frac{N_{ins}}{Time} = \frac{N_{ins}}{N_{clock}} \times \frac{N_{clock}}{Time}$$

The second factor indicates the frequency, an index of purely technological nature that has reached a saturation situation due to the physical limitations of hardware.

In fact the chip power efficiency is no longer improving at historical rates. Up until now, Moore's Law improvements in photolithography techniques resulted in proportional reductions in dynamic power consumption per transistor and consequent improvements in clock frequency at the same level of power dissipation— a property referred to as

Dennard scaling. However, below 90 nm, the static power dissipation (power lost due to current leakage through the silicon substrate) has overtaken dynamic power dissipation. This leads to a stall in clock frequency improvements in order to stay within practical thermal power dissipation limits. Thus, the free ride of clock frequency and power efficiency improvements is over.

It is therefore necessary, in order to improve energy efficiency to focus the attention on the study of the first factor of the product mentioned above, and that is the average number of instructions executed per clock cycle (IPC). This latter represents, a measure, not of how fast can go the circuit realization of our processor, but how well the architecture is able to execute instructions.

This translates into a greater attention to aspects such as software scalability, memory, IO, storage bandwidth and system resiliency to improve energy efficiency.

Paragraph 4.4: Power Trace of the application

In this context, it is inserted the following analysis, that is the study of the power consumption characteristics of the application in question.

Regarding the energy consumption of cluster Cavium's Thunderx, it presents the pretty badly characteristics and one of the reason is that power management either did not work, or at least did not work very well. Changing the power governor was not possible: the cpufreq driver was not recognized. The difference between peak and idle (+/- 80W) makes suspect that the chip is consuming between 40 and 50W at idle. Whether is just a matter of software support or a real lack of good hardware power management is not clear. It is quite possibly both.

The device used for the acquisition of energy consumption values and the evaluation of the power efficiency is the *Yokogawa Power Meter*.

It collects power data (V, W, A) from the 4 Thunder nodes since they only use one power supply for all of them, and it provides a serial interface where you can connect and, by sending specific information, receive the instant power data collected at the moment you query the Yokogawa.

During the initial setup, some changes were made to the job script, that is:

- Before your application starts its execution, the HCA server must have time to begin collecting data. This has been made possible by determining a phase of start and stop of the program.
- Since all energy consumption data are collected instantly on all 4 nodes, is required the allocation of 4 nodes for job, even in the case that serves only one node.
- It is necessary to collect the information on power consumption in a time frame from about 10 seconds before the application begins its execution at about 10 seconds after the execution ends, in order to compare the consumption during the execution with the state of idle of the cluster.

```
#!/bin/bash

#SBATCH -t 30:00
#SBATCH --ntasks=16
#SBATCH --cpus-per-task=1
#SBATCH -o powerTrace-%j.out
#SBATCH -e powerTrace-%j.err
#SBATCH -J powerTrace
#SBATCH --partition=thunder

POWER_TRACE_SCRIPT="/home/scc16/00-POWER_TRACE/getYokogawaMeasurements.py"
POWER_TRACE_FILE="/home/dbasciano/Cluster-3-Parallel_application/Measurements/measures.csv"
yourBinary="/home/dbasciano/Cluster-3-Parallel_application_nproc4"

# Start the power monitoring
ssh -f -v hca "$POWER_TRACE_SCRIPT $POWER_TRACE_FILE"

# Wait a bit until start your application
sleep 10s

## Run the application
## if only one node of the four of them
srun --cpus-per-task=1 --nodes=1 --ntasks=16 ./Project4DVAR
## if wanted to run on two nodes
#srun --cpus-per-task=1 --nodes=2 --ntasks=16 $yourAwesomeBinary

# wait a bit more until your awesome application ends
sleep 10s

# Stop the power monitoring
ssh hca "pkill -f -INT getYokogawaMeasurements.py"
```

The study conducted in this thesis work has focused mainly on an analysis of the energy consumption of the application by changing the number of running processes.

It was then examined a configuration with $\text{ppn} = 1$, $\text{ppn} = 16$ and a configuration with $\text{ppn} = 32$, mediated in order to minimize the uncertainty of the measurements, in order to "quantify":

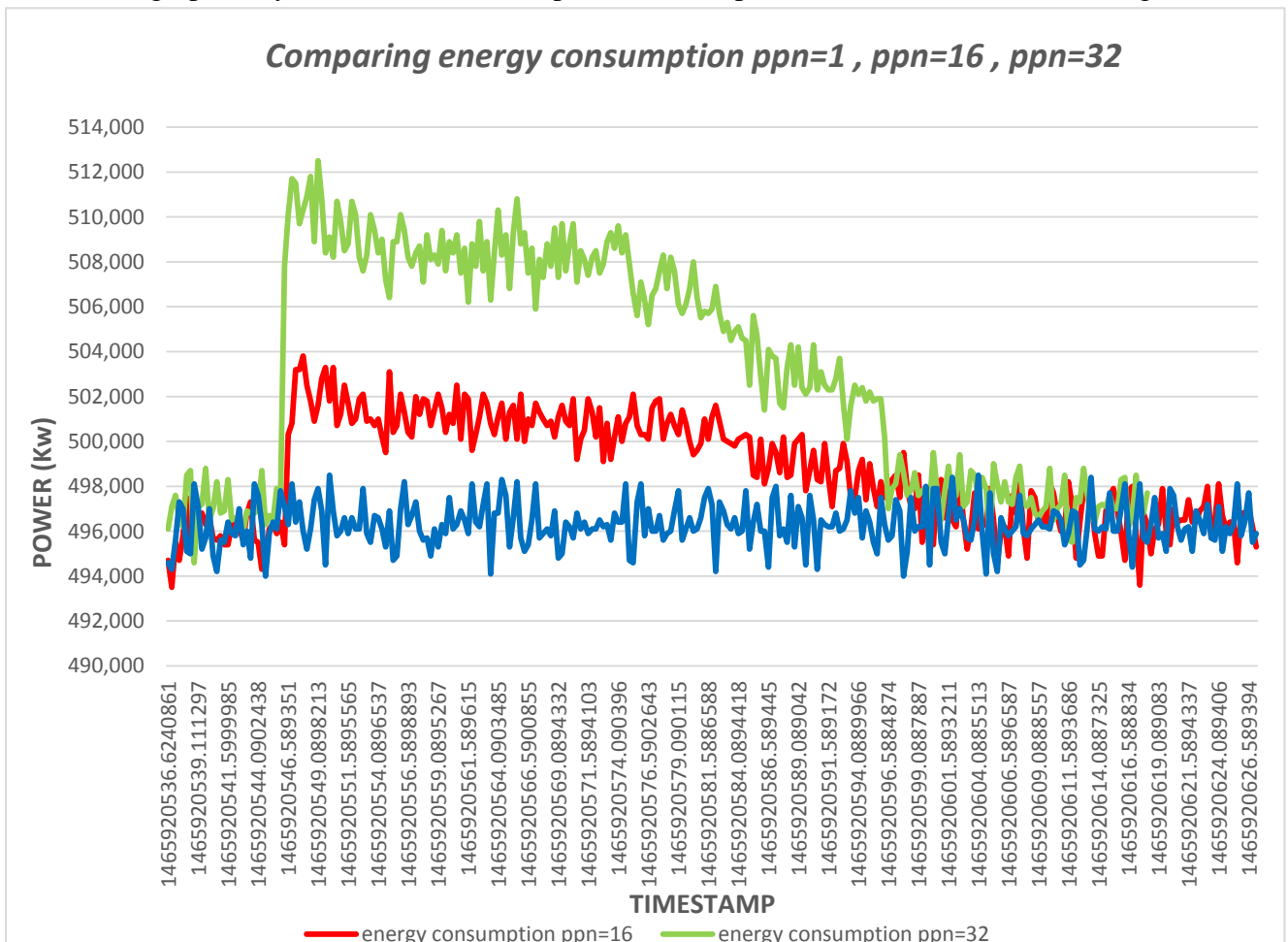
- **Idle power:** is defined as the power used by the system when it is not running a workload, but it is in a state where it is ready to accept a workload.

The idle state is not a sleep or a hibernation state. The idle measurement need not be made just before or after the workload is run.

Think of the idle power measurement as a constant of the system when no workload is running.

- **Power at execution time:** Energy consumption of the application during the execution time.

This choice of examining these specific cases is due to the need to mark a difference graphically between the various power consumptions. The result is the following:



Is possible to observe how the consumer with $\text{ppn}=1$ is very similar to the idle state, explainable in part with a light use of cluster.

Another observation that can be done concerns the energy difference between the case $\text{ppn}=1$ and the case $\text{ppn}=16$ estimable equal to about 5Kw as well as the energy difference between the case $\text{ppn}=16$ and the case $\text{ppn}=32$.

As can be noted, the power efficiency trends in cases $\text{ppn} = 16$ and $\text{ppn} = 32$, do not return to an idle power consumption in clean way. This can be attributed to the fact that the processes, inside one execution, are running a different number of iterations of the minimization loop before ending.

These energy considerations represent an application baseline, a starting point from which can then make a comparison with other programming styles such as OpenMP or an hybrid approach MPI/OpenMP.

Chapter 5: Inside Paraver Tool

As it is known, to write parallel applications that make a good use of resources is not an easy task.

A series of diagnostic tools support the developer in the evaluation phase and profiling, and in the phases of tuning and optimization of the code.

The performance tools developed at BSC are an open-source project targeting not only to detect performance problems but to understand the application's behavior.

The key component that has allowed me to go into the details of my application is the tool Paraver.

Paraver was developed to respond to the need to have a qualitative global perception of the application behaviour by visual inspection and then to be able to focus on the detailed quantitative analysis of the problems.

Performance information in Paraver is presented with two main displays that provide qualitatively different types of information.

The timeline display represents the behaviour of the application along time and processes, in a way that easily conveys to the user, a general understanding of the application behaviour and simple identification of phases and patterns.

The statistics display provides numerical analysis of the data that can be applied to any user selected region, helping to draw conclusions on where and how to focus the optimization effort.

This chapter is structured in this way:

- In the first paragraph, it was described the process of extraction of a trace from the application with the use of tool *Extræe*.
- In the second paragraph, it was introduced the analisys of the application with the Paraver tool.

Paragraph 5.1: Extræe

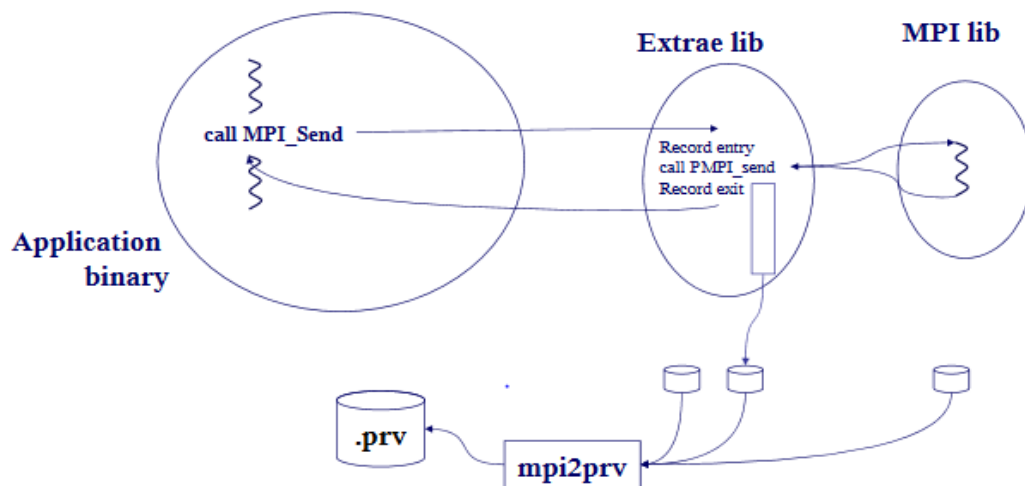
Extræe is the package devoted to generate Paraver trace-files for a post-mortem analysis.

Extræe is a tool that uses different interposition mechanisms to inject probes into the target application so as to gather information regarding the application performance.

Extræe takes advantage of multiple interposition mechanisms to add monitors into the application

The interposition mechanism used in this case is Linker preload (LD_PRELOAD) to intercept production binaries at loading time.

The instrumentation process can be summarized in this way:



The information collected by Extræe includes entry and exit to the programming model runtime (MPI calls), hardware counters (PAPI), call stack reference, user functions, periodic samples and user events (API).

In order to facilitate the configuration, Extrae can be configured through an XML file that exemplifies all the options available to set up in the configuration file.

Most of the nodes present in the XML file have an enabled attribute that allows turning on and off some parts of the instrumentation mechanism.

Paragraph 5.1.1: Extraction process of a trace

In order to extract a track from your application, is necessary to instrument the code with the following changes:

- compile the application using the following flag: ***-funwind-tables -g***
- copy in the current path the necessary files to the extraction process, that is the configuration file ***extrae.xml*** and the file ***trace.sh***
- customize the XML file ***extrae.xml*** according to your needs (for example to activate MPI instrumentation). An example is the following:

```
<?xml version='1.0'?>
<trace enabled="yes"|
home="/apps/extrae/3.2.1/opnmpi/1.10.2"
initial-mode="detail"
type="paraver"
xml-parser-id="Id: xml-parse.c 3465 2015-10-19 10:14:17Z harald $"
>

<mpi enabled="yes">
  <counters enabled="yes" />
</mpi>

<openmp enabled="yes">
  <locks enabled="no" />
  <counters enabled="yes" />
</openmp>

<pthread enabled="no">
  <locks enabled="no" />
  <counters enabled="yes" />
</pthread>

<callers enabled="yes">
  <mpi enabled="yes">1-3</mpi>
  <sampling enabled="no">1-5</sampling>
  <dynamic-memory enabled="no">1-3</dynamic-memory>
</callers>

<user-functions enabled="yes" exclude-automatic-functions="no">
  <counters enabled="yes" />
</user-functions>
```

```

<counters enabled="yes">
  <cpu enabled="yes" starting-set-distribution="1">
    <set enabled="yes" domain="all" changeat-time="0">
      PAPI_TOT_INS,PAPI_TOT_CYC,PAPI_L2_DCM,PAPI_L2_TCM,PAPI_L2_ICM,PAPI_L2_LDM,PAPI_FP_INS
    </set>
    <set enabled="yes" domain="all" changeat-time="0">
      PAPI_TOT_INS,PAPI_TOT_CYC,PAPI_LD_INS,PAPI_SR_INS,PAPI_BR_UCN,PAPI_BR_CN,PAPI_VEC_SP,RESOURCE_STALLS
    <sampling enabled="no" period="1000000000">PAPI_TOT_CYC</sampling>
    </set>
  </cpu>
  <network enabled="no" />
  <resource-usage enabled="no" />
  <memory-usage enabled="no" />
</counters>

<storage enabled="no">
  <trace-prefix enabled="yes">TRACE</trace-prefix>
  <size enabled="no">5</size>
  <temporal-directory enabled="yes">/scratch</temporal-directory>
  <final-directory enabled="yes">/gpfs/scratch/bsc41/bsc41273</final-directory>
  <gather-mpits enabled="no" />
</storage>

<buffer enabled="yes">
  <size enabled="yes">500000</size>
  <circular enabled="no" />
</buffer>

<trace-control enabled="no">
  <file enabled="no" frequency="5M">/gpfs/scratch/bsc41/bsc41273/control</file>
  <global-ops enabled="no"></global-ops>
  <remote-control enabled="no">
    <signal enabled="no" which="USR1"/>
  </remote-control>
</trace-control>

<others enabled="no">
  <minimum-time enabled="no">10M</minimum-time>
</others>

<bursts enabled="no">
  <threshold enabled="yes">500u</threshold>
  <mpi-statistics enabled="yes" />
</bursts>

<sampling enabled="no" type="default" period="50m" variability="10m" />

<dynamic-memory enabled="no">
  <alloc enabled="yes" threshold="32768" />
  <free enabled="yes" />
</dynamic-memory>

<input-output enabled="no" />

<merge enabled="yes"
  synchronization="default"
  tree-fan-out="16"
  max-memory="512"
  joint-states="yes"
  keep-mpits="yes"
  sort-addresses="yes"
  overwrite="yes"
/>

</trace>

```

- edit the trace wrapper *trace.sh* uncommenting the line related to the interposition mechanism LD_PRELOAD for C apps
- add the new wrapper in the job submission script, in the line where there is the *srun* command.

When the application ends, the trace generated consists of three files: *name_executable*.prv*, *name_executable*.pcf*, *name_executable*.row*.

The file with extension *.prv* contains the trace records classified in these following fields:

- *Record Type*: state, event, communication.
- *Context*: application : process : thread
- *CPU*
- *Event*: t_{event} : type: value
- *State*: t_{start} : t_{end} : state
- *Comm Source/Comm Dest with size and tag*

Paragraph 5.2: Analysis with Paraver

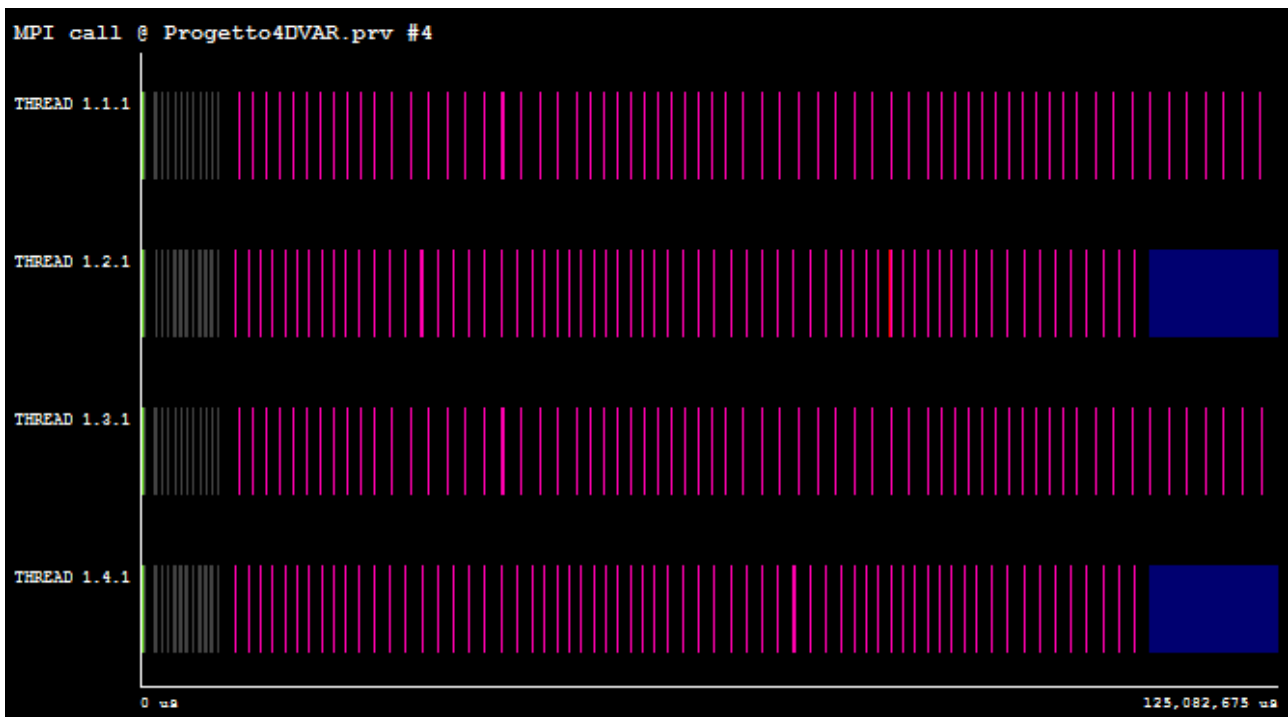
The analysis carried out with the Paraver tool allows to study in detail the style of parallelization adopted in order to highlight any incorrect behavior, unbalancing of load or any misalignment in the execution of the various MPI calls.

The tuning step has been focused, initially on the case of the parallel version with $ppn = 4$ and in the second place on the version with $ppn = 16$.

Paragraph 5.2.1: Parallel version with $ppn=4$

The first step was to load into the tool the track executable **.prv* obtained.

The first configuration considered is the *MPI_Call.cfg*: it exhibits a temporal vision of the various MPI calls, distinct because of a different color gradation, performed by the various processes.



Pointing the mouse on a specific color we can determine the specific type of MPI call, and we may notice also its duration in (us) in the Info panel.

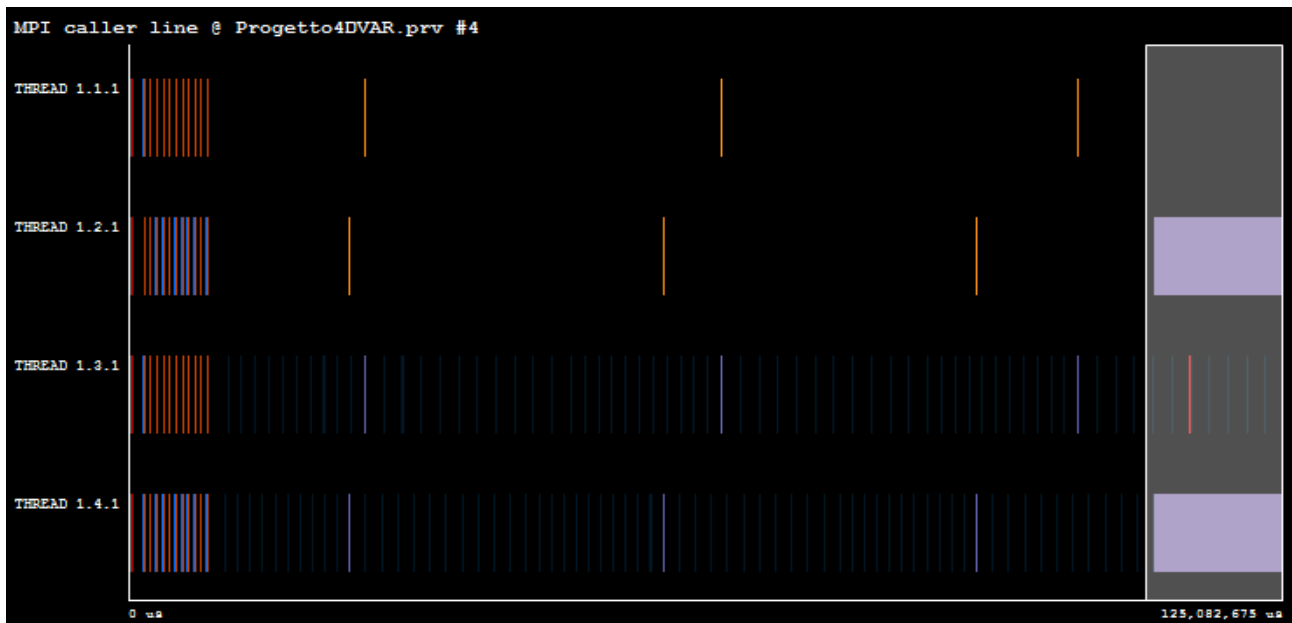
At this granularity you can see that the threads 2 and 4 end before their parallel code portion, and put themselves on hold in MPI_Finalize for about 14 seconds waiting for the end of the other two threads.

This situation may be due, for example, to an imbalance of the load between the various worker, which entails the need to perform an additional iteration of the minimization loop to terminate execution of the workers 1 and 3.

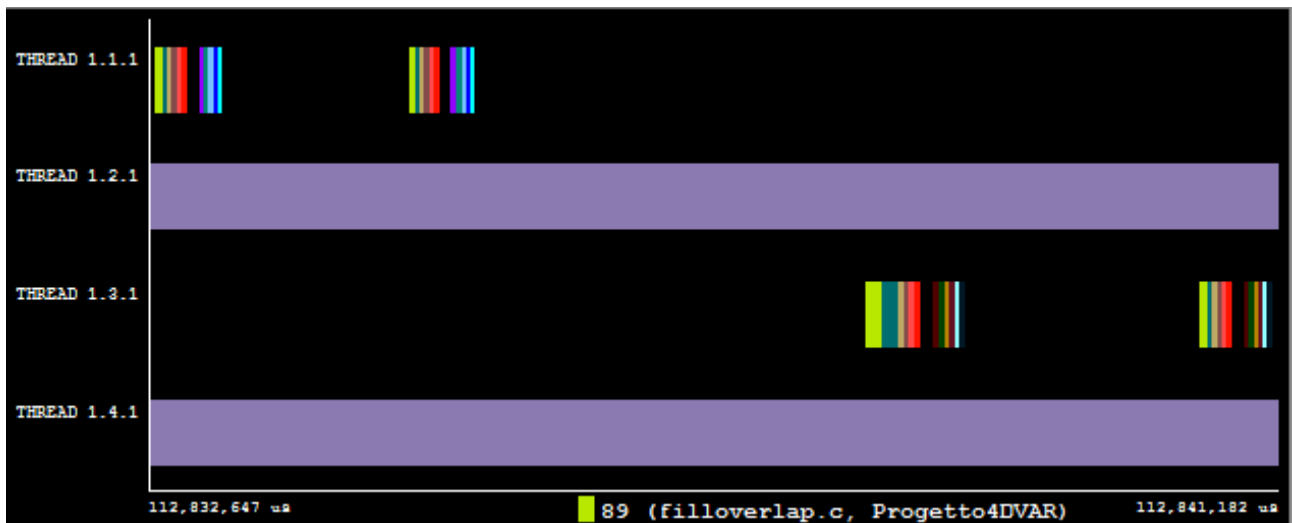
To examine in detail this situation, the tool has an additional configurations:

- ***MPI Caller line***, who shows on which line of code reside in all times the various workers, thus highlighting how it is necessary for the workers 1 and 3 re-run the various instructions of the loop.

In fact highlighting the MPI region of interest:



And by carrying out a zoom into the selected area, the two workers are still performing MPI communications in *filloverlap.c* module:



Other characteristic is the MPI message size of point-to-point calls, exchanged between the various workers in the Send and Receive operations.

This is computed by configuration file *3dh_msgsize_per_pt2pt_call.cfg*.

A typical concern when an MPI program does not scale is that it may be using many small messages.

For many reasons could be interested in finding which message sizes are used by the

application.

Instead, configuration file *total_bw.cfg* can be used to visualize the instantaneous amount of communication bandwidth used by point to point calls in the application.

Some interpretation of this configurations could be:

- The metric *total_bandwidth* quantifies the equivalent bandwidth a network should have provided not to delay the execution. It is not the actual network bandwidth of the real system (for example a load imbalance in the application may result in a low bandwidth as there may be a lot of time available for the transfer, even if the real system performs a very fast transfer between processors and then the message has to wait for the reception to arrive). The metric is an indication of what the application demands or is able to achieve in the real system. In regions where the demand is very high the metric will saturate if the limit of the network is reached
- Useful to identify regions where the network bandwidth may actually be limiting the application performance

This type of analysis allows also me to quantify the transmission band of my application and then the opportunity to draw a graph buffer size / execution time.

A way to determine the global efficiency of the application is to use the configuration file *General/analysis/avg_procs.cfg*: a single entry 2D window reporting the average number of processes performing useful computation out of the total number of processes.

A plausible interpretation of this configuration may be to determine a number which reflects the efficiency on the application in terms of its parallel execution.

Ideally, the value should be as close as possible to the total number of processors for good performance, the higher number of CPUs kept active working on computational parts of the problem.

In this case the configuration returns a value equal to **3.84**.

The configuration *efficiency.cfg* shows a statistic about the percentage of efficiency of each thread:

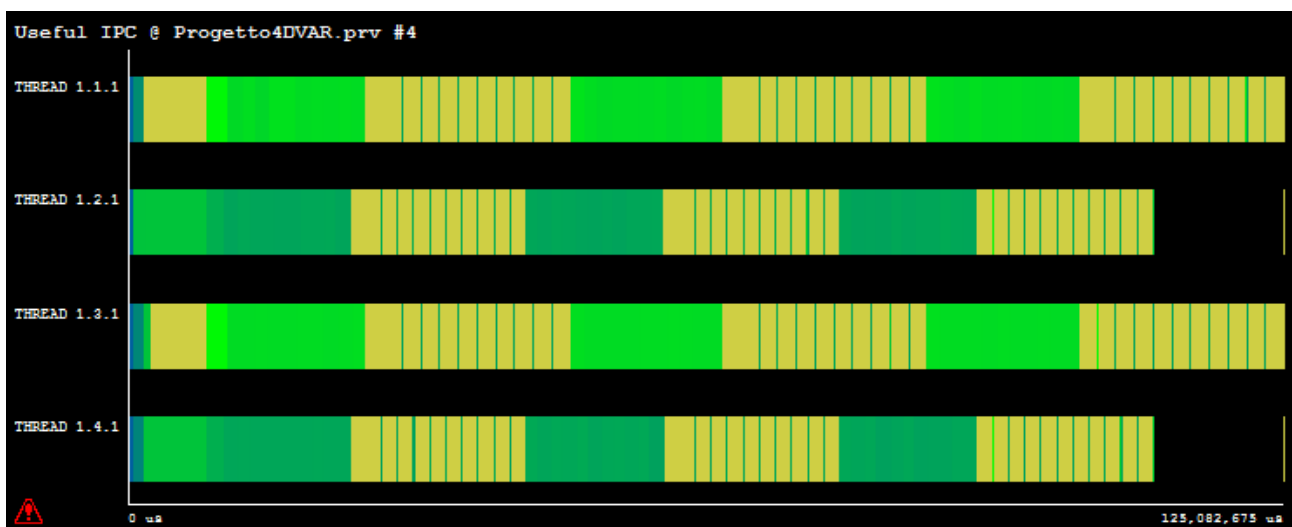
	Idle	Running
THREAD 1.1.1	0.25 %	99.75 %
THREAD 1.2.1	12.56 %	87.44 %
THREAD 1.3.1	0.32 %	99.68 %
THREAD 1.4.1	12.63 %	87.37 %
Total	25.76 %	374.24 %
Average	6.44 %	93.56 %
Maximum	12.63 %	99.75 %
Minimum	0.25 %	87.37 %
StDev	6.16 %	6.16 %
Avg/Max	0.51	0.94

Another examined configuration is the `useful_IPC.cfg`: it shows a timeline with the *IPC* parameter (instruction per cycle) = $(instr / cycles) * useful$ obtained in certain computational intervals.

Selecting the Info Panel by right clicking and selecting the Tab color is possible to observe the representative scheme with various shades of colors with their associated values of IPC.

The IPC function, in fact, for each process can have a gradation ranging from light green, representing a low value of IPC, to a dark blue, representing a high IPC value.

The result obtained is the following:



The maximum level of the IPC reached in the application is equal to a value of 0.71 obtained by the processes 2 and 4.

The analysis previously performed is directly designed on a time scale, a less detailed statistical perspective in some cases can often be enough to identify problems and give an overview of the behavior of an application.

Paraver provides a mechanism, called *2D Analyzer*, to obtain some profiles for specific regions of a track, in the form of tables with the summary statistics.

An interesting configuration is the *MPI_Stats*: it builds a table that presents a row for each thread and a column for each MPI call.

The first column corresponds to the time out of the MPI region. Each entry in the table tells the percentage of time that the corresponding thread spent in the MPI function.

Clicking on the magnifying glass it is possible to see the following numerical values:

	Outside MPI	MPI_Isend	MPI_Irecv	MPI_Scatterv	MPI_Comm_rank
THREAD 1.1.1	99.83 %	0.03 %	0.03 %	0.10 %	0.00 %
THREAD 1.2.1	98.70 %	0.03 %	0.03 %	1.24 %	0.00 %
THREAD 1.3.1	99.75 %	0.03 %	0.03 %	0.19 %	0.00 %
THREAD 1.4.1	98.61 %	0.03 %	0.03 %	1.34 %	0.00 %
Total	396.89 %	0.13 %	0.12 %	2.87 %	0.00 %
Average	99.22 %	0.03 %	0.03 %	0.72 %	0.00 %
Maximum	99.83 %	0.03 %	0.03 %	1.34 %	0.00 %
Minimum	98.61 %	0.03 %	0.03 %	0.10 %	0.00 %
StDev	0.57 %	0.00 %	0.00 %	0.57 %	0.00 %
Avg/Max	0.99	0.93	0.85	0.54	0.94

Changing metric with the Statistic Selector, for example:

- The option Burst that counts the number of occurrences of each MPI call,
- The option Burst Time Average: that is the average length of a MPI call,

It is possible to see a different statistic:

	Outside MPI	MPI_Isend	MPI_Irecv	MPI_Scatterv	MPI_Comm_rank
THREAD 1.1.1	53,673.18 us	32.22 us	31.67 us	40,966.95 us	55.96 us
THREAD 1.2.1	58,240.12 us	41.36 us	30.30 us	15,420.52 us	56.52 us
THREAD 1.3.1	57,020.65 us	36.19 us	37.59 us	3,277.72 us	56.04 us
THREAD 1.4.1	51,223.01 us	38.14 us	27.32 us	38,930.16 us	55.73 us
Total	220,156.96 us	147.91 us	126.90 us	98,595.35 us	224.26 us
Average	55,039.24 us	36.98 us	31.72 us	24,648.84 us	56.06 us
Maximum	58,240.12 us	41.36 us	37.59 us	40,966.95 us	56.52 us
Minimum	51,223.01 us	32.22 us	27.32 us	3,277.72 us	55.73 us
StDev	2,765.93 us	3.31 us	3.74 us	15,906.94 us	0.29 us
Avg/Max	0.95	0.89	0.84	0.60	0.99

Finally, it focused attention on some metrics derived from the events based on Hardware counter that are generated in the trace, at each input or output from MPI functions.

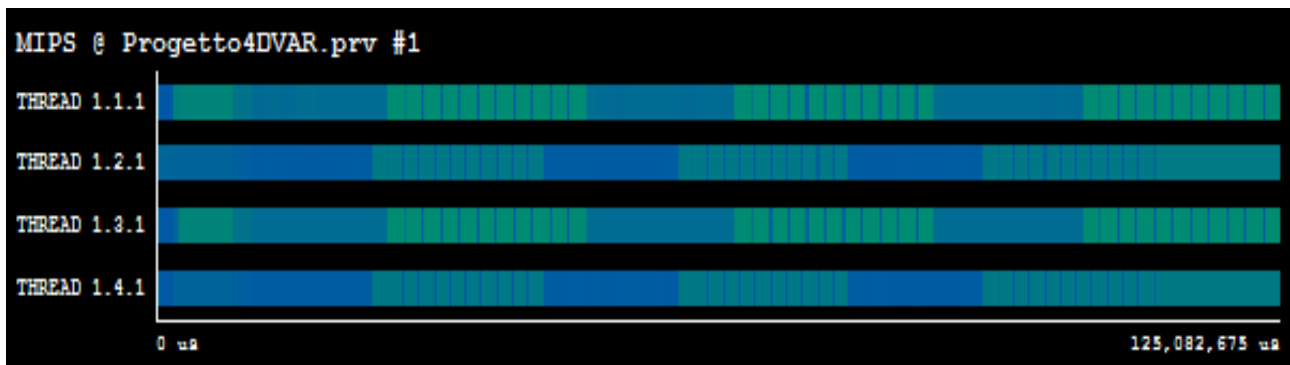
The views may be directly obtained from a single hardware counter or be derived metrics combining several of them. Each view represents a time varying function typically color encoded

Among the metrics counters, we have the Counters_PAPI configurations grouped into 4 main categories:

- Program: evaluation of application performance, and therefore not dependent on the specific used platform,
- Architecture: related to execution on specific architectures (i.e. cache misses...),
- Performance: metrics reporting rates per time (i.e. MFLOps, MIPS, IPC...)
- Models: This section contains configuration files where a simple model of what the IPC (nstructions epr cycle) or elapsed time should be for each interval between to samples as a function of the acquired hardware counts. In general they could/should be compared to the measured IPC or elapsed time also available from the trace.

About the performance metric, we obtain with the configurations *MFlops.cfg* and *MIPS.cfg* the following graphs:





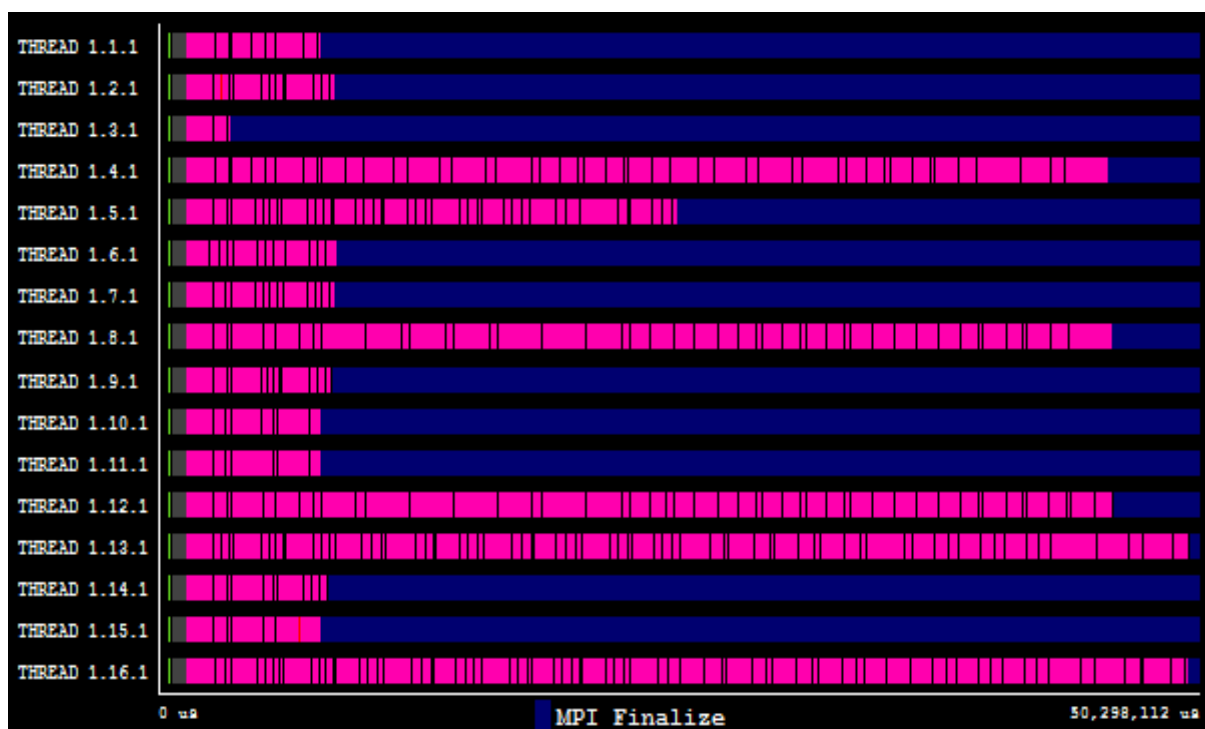
The first give the estimate of mega floating-point operations per second, MFLOPs, equals to the maximum of **152,440**, a common measure of the speed of computers used to perform floating-point calculations.

The second graph, instead, returns the estimate of MIPS (million instructions per second) in a maximum range from **1,266.55** to **1,271.68**, another common measure of computer speed and power.

Paragraph 5.2.2: Parallel version with ppn=16

The same path of analysis has concerned the case with ppn = 16, which presents a larger misalignment between the operations of the various thread.

In fact, the configuration *mpi_call.cfg* returns the following display:

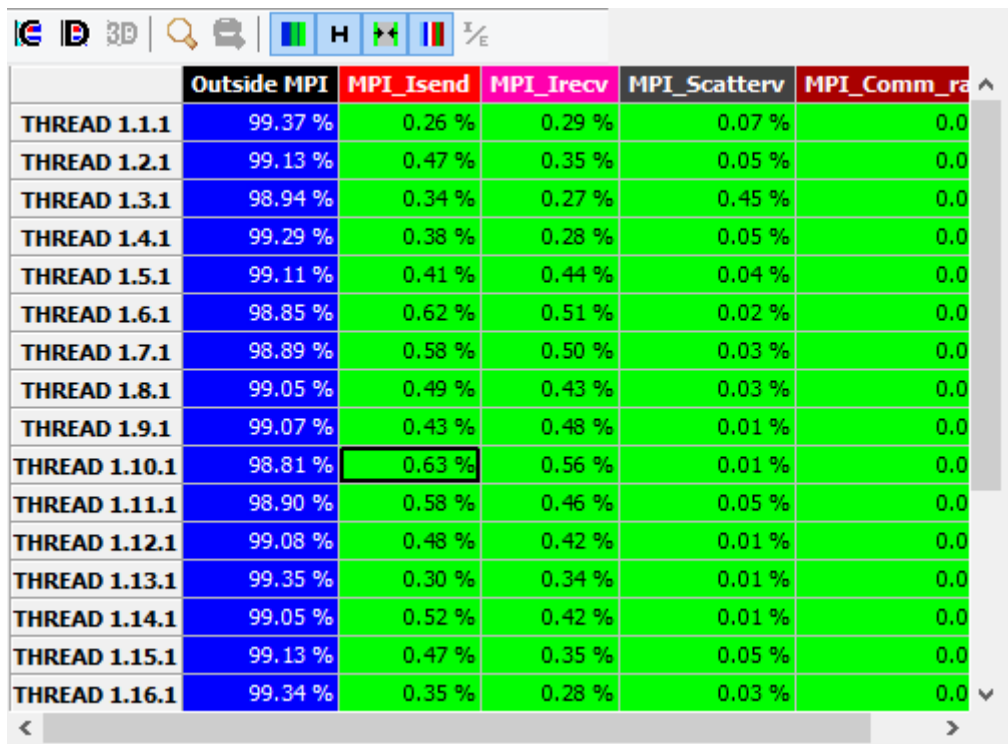


where, due to a load of different work among the various workers, some thread terminate before their minimization process and wait in `MPI_Finalize()` state.

When talking about different workload between the worker, it is not intended a different amount of data but a different range of values within those data.

This difference in values is reflected on a different number of iterations required to stop the minimization algorithm

Is possible to see, in effect, with the configuration *MPI_stats.cfg* how the worker 10 spends more time than others in the various `MPI_Send()` and `MPI_Recv()`:



	Outside MPI	MPI_Isend	MPI_Irecv	MPI_Scatterv	MPI_Comm_rank
THREAD 1.1.1	99.37 %	0.26 %	0.29 %	0.07 %	0.0
THREAD 1.2.1	99.13 %	0.47 %	0.35 %	0.05 %	0.0
THREAD 1.3.1	98.94 %	0.34 %	0.27 %	0.45 %	0.0
THREAD 1.4.1	99.29 %	0.38 %	0.28 %	0.05 %	0.0
THREAD 1.5.1	99.11 %	0.41 %	0.44 %	0.04 %	0.0
THREAD 1.6.1	98.85 %	0.62 %	0.51 %	0.02 %	0.0
THREAD 1.7.1	98.89 %	0.58 %	0.50 %	0.03 %	0.0
THREAD 1.8.1	99.05 %	0.49 %	0.43 %	0.03 %	0.0
THREAD 1.9.1	99.07 %	0.43 %	0.48 %	0.01 %	0.0
THREAD 1.10.1	98.81 %	0.63 %	0.56 %	0.01 %	0.0
THREAD 1.11.1	98.90 %	0.58 %	0.46 %	0.05 %	0.0
THREAD 1.12.1	99.08 %	0.48 %	0.42 %	0.01 %	0.0
THREAD 1.13.1	99.35 %	0.30 %	0.34 %	0.01 %	0.0
THREAD 1.14.1	99.05 %	0.52 %	0.42 %	0.01 %	0.0
THREAD 1.15.1	99.13 %	0.47 %	0.35 %	0.05 %	0.0
THREAD 1.16.1	99.34 %	0.35 %	0.28 %	0.03 %	0.0

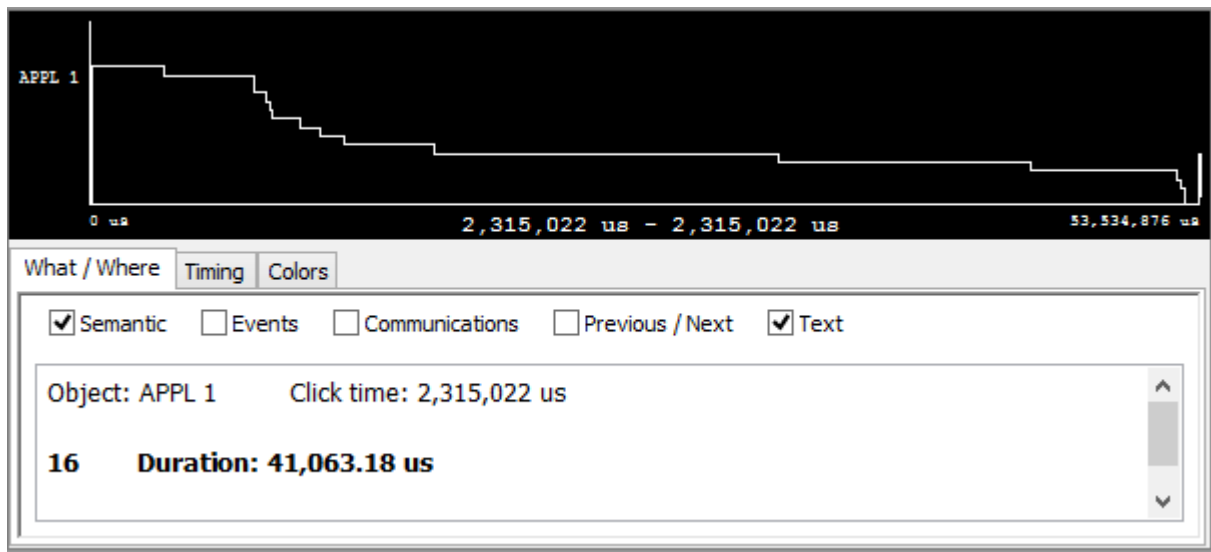
content is the communication a real bottleneck for the application performance.

Some interpretation capabilities can be:

- Large values in one MPI call may draw our attention to it. Before thinking of modifying the structure of the source code it might be interesting to further investigate the performance of the MPI calls
- Variations across processes in column 0 probably indicate computational load imbalances.

Waits for message reception due to imbalances or externally caused delays (i.e. preemptions) that propagate through the communication dependence chain.

An other configuration file *General/views/instantaneous_parallelism.cfg* displays the total number of processes performing some useful computation at each point in time. This view will point out to regions of poor performance. Ideally a user would like its application to have a constant number of active processors equal to the allocated number of processors. Regions of low value of this metric should probably be analysed as occurs in this case:



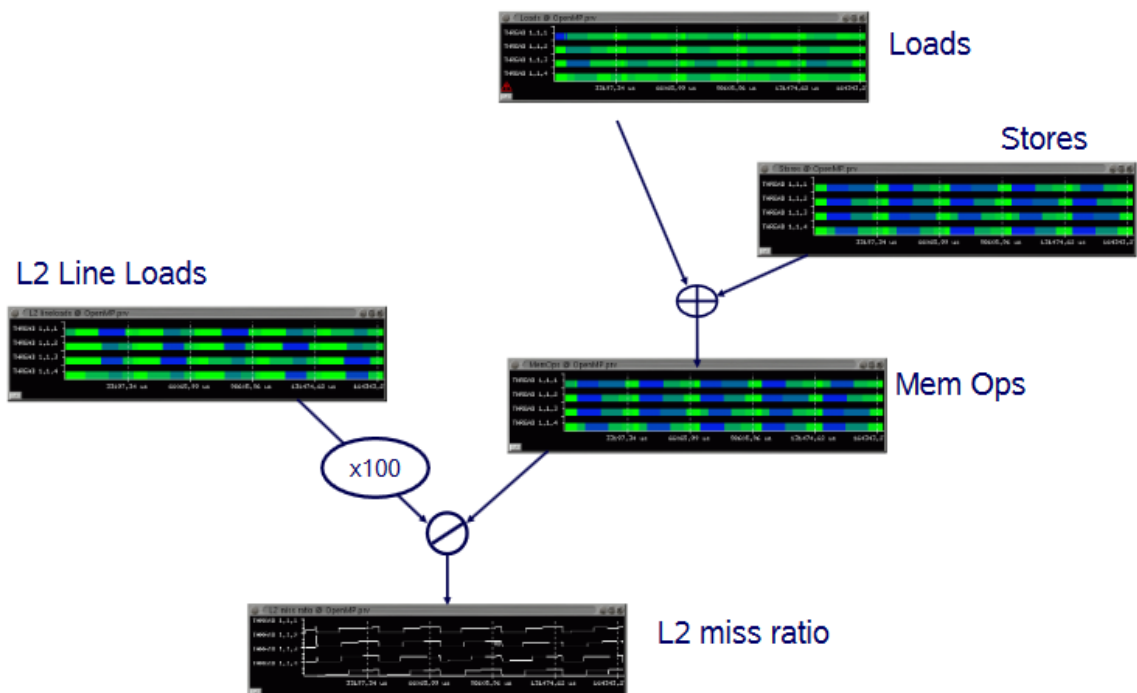
Regarding the global efficiency of the application, the configuration file *General/analysis/avg_procs.cfg* reports the average number of processes performing useful computation out of the total number of processes, a value equal to **10.65**.

This result is mainly due to the waiting time of specific thread in the state of `MPI_Finalize()` and then to a little their synchronous termination.

Paragraph 5.2.3: Vertical deepening of the cache management

Other very interesting perspective of analysis that allows to also explain the reasons for the super linear speedup mentioned above, is the configuration named *L1_missratio.cfg*, and *L2d_miss_ratio.cfg* related to Architecture metrics, and obtained with the following

path:



Before examining the graph result from the following configuration is appropriate to describe a small overview of Thunder chip: the Cavium uses a custom ARM core, which has 78 KB of instruction cache and 32 KB of data cache per core and 16 MB of L2 cache that is shared across all of the cores on the die.

The 37-way 78 KB L1 cache is certainly odd, but it might be more than just "network processor heritage". Few academic studies have shown that scale-out workloads such as memcached have a higher than normal cache miss rate.

A reason why we believe Cavium has done its homework, is the fact that more die area is spent on cores (up to 48) than on large caches; an L3 cache is nowhere to be found. The Thunder-X has only one centralized relatively low latency 16MB L2 cache running at full core speed. A lot of academic studies have confirmed that a large L3 cache is a waste of transistors for scale-out workloads.

In other words, an L3 cache just adds more latency to requests that missed the L1 cache and that will end up in the DRAM anyway. That is also the reason why Cavium made sure that a beefy memory controller is available: the Thunder-X comes with four DDR3/4 72-bit memory controllers and it currently supports the fastest DRAM available

for servers.

The study was conducted by analyzing the maximum amount of cache misses ratio (L2 cache misses per 1000 instructions) in various cases on mini-cluster Thunder, with the purpose to valorize the results obtained in the field of performance.

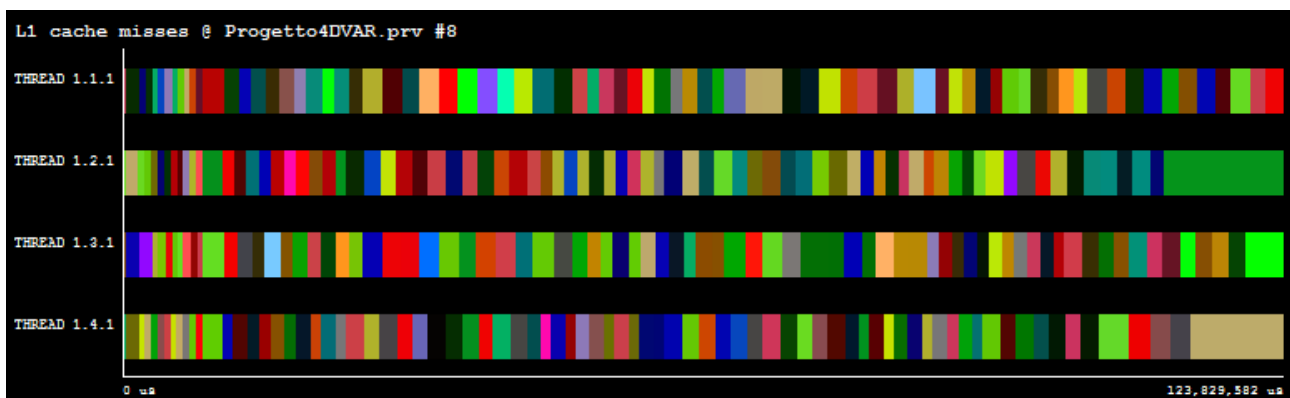
How is possible to see from the following table:

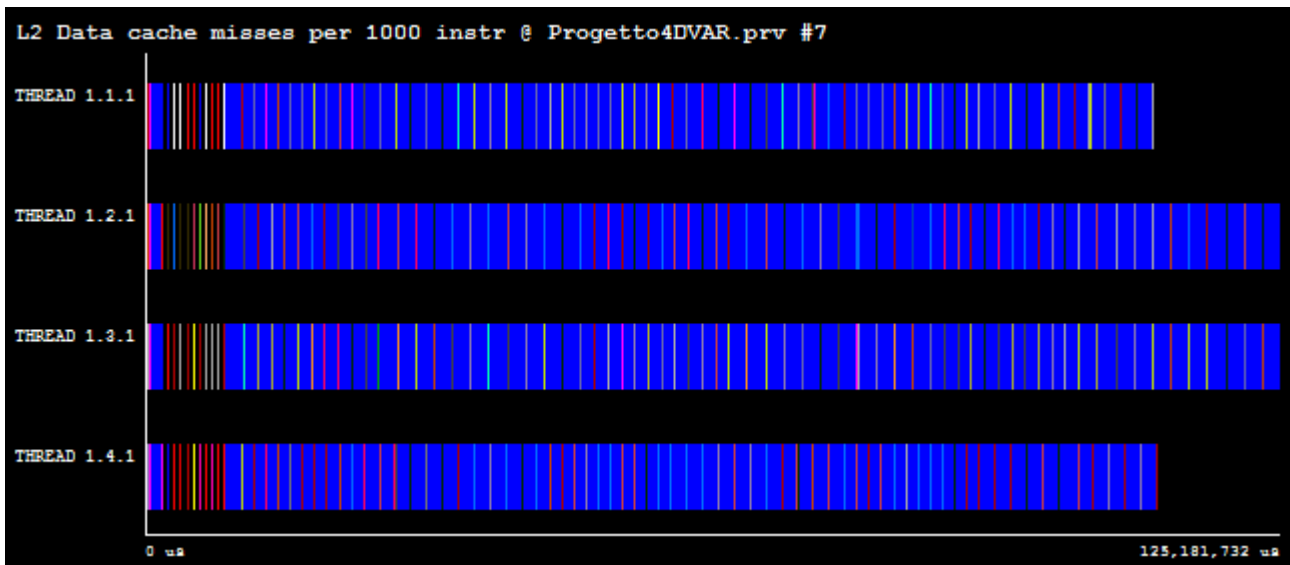
Process per node	Maximum cache miss ratio value	Cache miss / process per node
1	97,234	97,234
2	54,588	27,294
4	33,244	8,311
8	63,781	7,9761
16	74,201	4,637

The case with a single process exhibits a very large value of cache miss, value that is gradually decreasing if mediated by the number of processes by emphasizing a more effective use of resources hardware.

Given this context, the configuration files mentioned above, show a timeline with the L1 cache miss and L2 cache miss ratio (L2 cache misses per 1000 instructions) in each interval between MPI events.

In the case $ppn = 4$, were obtained the following graph:

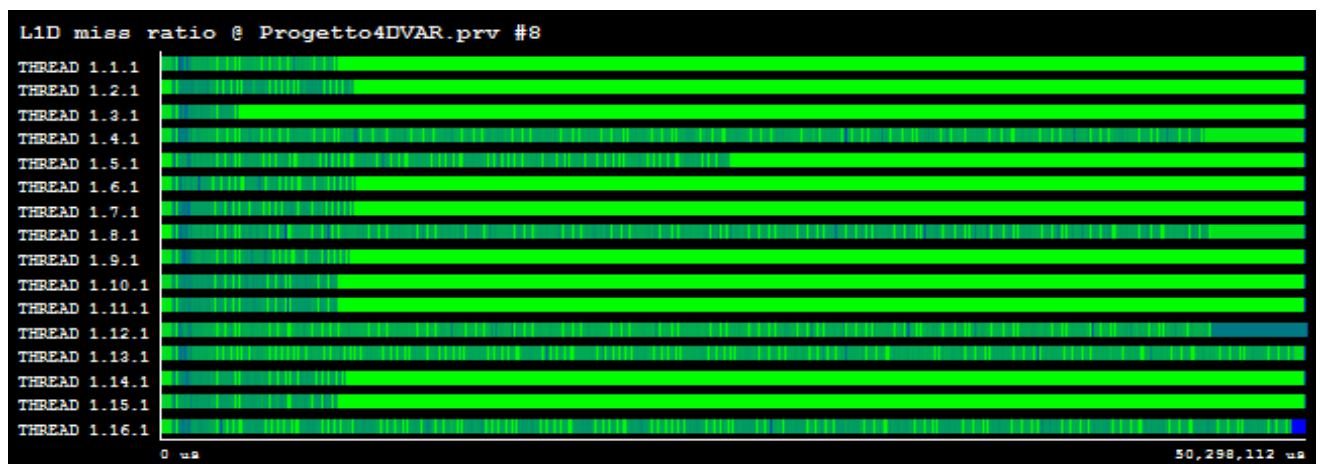




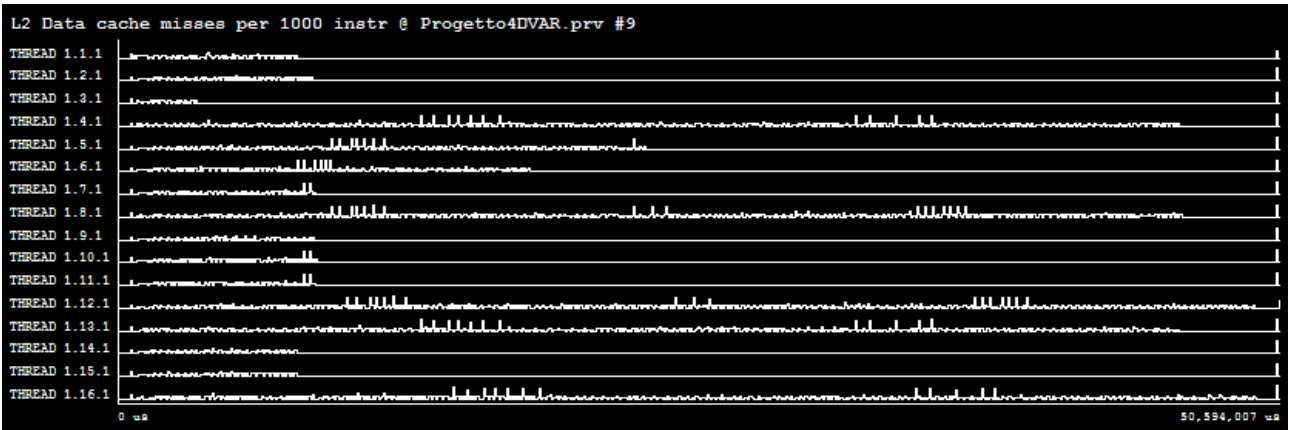
Displaying according to the gradation of color can be observed as the case of the L1 cache is much more “unstable” than in the case of the L2 cache.

As can be seen the two graphs refer to traces of two different executions of the case $ppn=4$, for the impossibility of being able to use more than 8 hardware counter in the configuration file *extrae.xml*.

The same configurations were applied to the case with 16 processes:



Viewing the graph of the percentage of L2 cache misses and clicking on the Function line mode can be identified, through an overlap with the graph MPI_Caller_Line, as peaks can be attributed to the calls of MPI_Recv and MPI_Send contained in filloverlap.c module.



Conclusioni e sviluppi futuri

The study of the application in this work of thesis has shown, in terms of execution times, as a parallelism of multiprocessor type goes well with the problem in question given the gains on the time scale.

On the other hand, it has also emerged that, in the case of the running on HCA server, the efficiency graphics do not return the expected values because of an unchanged size of the initial dataset. While, in the case of running on Thunder server, it has been showed how the phenomenon of super linear speedup can be explained by a more efficient use of the cache management.

In addition, the use of Paraver tool has enabled a thorough inspection of the code highlighting, as necessary, its revision in an effort to improve the parallelization performed.

The intent will be to make some improvements to the MPI version to try to obtain increased indices of performance and efficiency (IPC) value, though, for example, an excellent decomposition of the initial datasets between the various processes, on the basis of knowledge of the range of examined values.

In this way the processes would terminate at the same instant the loop of minimization avoiding situations of idle due to waiting and then increasing the processor usage percentage.

At the same time, using the present version as a baseline, an idea would be to develop a

OpenMP version of the application in order to introduce a form of multicore parallelization so as to make a comparison both in terms of performance both in terms of power efficiency on the mini-cluster Thunder.

Another future development that may bring benefits in terms of performance is the use of optimized mathematical libraries like ATLAS or math libraries for HPC applications on ARM platforms.

The last stage of this path may provide a porting of the application on the Mont-Blanc prototype in order to obtain a more accurate and detailed energy vision of the application, being able to have a set of finer granularity.

Bibliografia

- [1] Arcucci Rossella, “Tecniche numeriche per la risoluzione in ambiente parallelo del problema "Data Assimilation": un problema inverso mal posto”
- [2] <http://www.montblanc-project.eu/>.
- [3] Valgrind, <http://valgrind.org/>
- [4] Mantovani Filippo, “High Performance Computing based on mobile embedded processor”, EMiT 2015
- [5] Jing Shan, “Superlinear Speedup in Parallel Computation”
- [6] <http://www.top500.org/>
- [7] E. L. Padoin, P. O. A. Navaux, “High Performance Computing vs High Power Consumption”
- [8] <https://www.bsc.es/computer-sciences/performance-tools/paraver>
- [9] <https://www.bsc.es/computer-sciences/extrae>
- [10] <http://www.anandtech.com/show/10353/investigating-cavium-thunderx-48-arm-cores/>
- [11] R. Arcucci, L. D’Amore, Luisa Carracciuolo, “DD-OceanVar: A Domain Decomposition Fully Parallel Data Assimilation Software for the Mediterranean Forecasting System”, December 2013
- [12] <http://www.anandtech.com/show/8776/arm-challenging-intel-in-the-server-market-an-overview/4>
- [13] <https://www.bsc.es/computer-sciences/performance-tools/documentation/mpioopenmp-performance-analysis-tips>

- [14] R. S. Rejitha, “Energy consumption analysis and energy optimization techniques of HPC applications
- [15] https://www.bsc.es/sites/default/files/public/computer_science/extreme_computing/paraverpatc-oct.pdf